



A PRACTICAL CONTROL SYSTEM DESIGN PHILOSOPHY

Guidelines for Consistent, Maintainable, and Operator-Focused Control Systems

Version: 1.0

Status: Initial Open Release

Date: March 2025

This document is released under an open-source licensing model to encourage transparency, collaboration, and vendor-independent system design.

Contents

Licensing Agreement.....	6
Introduction	7
Warning	8
Benefits of Open Sourcing	9
Additional Points	9
Definitions	10
Equipment Overview	11
Argument for Adopting Modular Sequence-Based PLC Programming (MSBP)	12
1. Enhanced Fault-Finding and Troubleshooting Efficiency	12
2. Scalability and Future-Proofing.....	12
3. Cost Reduction and Control System Optimization	13
4. Improved Safety, Reliability, and System Control	13
5. Modular Programming for Faster System Updates and Maintenance	13
6. Streamlined Operator and Technician Training	14
7. Reduced Risk of Programming Errors and System Failures.....	14
Conclusion: A More Efficient and Reliable PLC Programming Method	14
Safety Instrumented Systems (SIS) – Scope and Exclusions	16
How It Works	18
What are Sub Systems?.....	19
What are Systems?	21
What are Assemblies?.....	23
What are Sequences?	24
What are Stages	26
Standby (Index: 0000 → 0999).....	28
Starting (Index: 1000 → 1999).....	28
Started (Index: 2000 → 2999)	28
Warm-up (Index: 3000 → 3999)	28
Running [Initial] (Index: 4000 → 4999)	29
Running [Manual] (Index: 5000 → 5999).....	29
Running [Auto] (Index: 6000 → 6999)	29
Warm-Down (Index: 7000 → 7999)	29
Stopping (Index: 8000 → 8999).....	30

Stopped (Index: 9000 → 9499).....	30
Operator Intention (Index: 9500 → 9899)	30
System Fault (Index: 9900 → 9998)	30
Emergency Stop (Index: 9999)	32
Programming Main Sequence & System Sequence(s)	33
Modification Notation	33
Input Section.....	34
Basic Digital Transfer	35
Basic Analog Transfer	37
Scaling Inputs.....	38
HMI Input Filtering.....	39
Default Parameter Restoration Section	40
General Operation Section	41
Auto Run Function	42
Fault Section.....	43
Time Monitoring	44
Low Level Monitoring (with Nuisance Protection).....	45
High Level Monitoring (with Nuisance Protection)	46
Range Monitoring (with Nuisance Protection)	47
Contactor Monitoring (with Nuisance Protection)	48
Fault Signalling (Immediate Shutdown).....	49
Fault Signalling (Controlled Shutdown).....	50
PID Section.....	51
Sequence Section.....	52
STANDBY (Index: 0000)	53
STARTING (Index: 1000)	54
STARTED (Index: 2000).....	55
WARM-UP (Index: 3000).....	56
RUNNING ► INITIAL (Index: 4000).....	57
RUNNING ► MANUAL (Index: 5000).....	60
RUNNING ► AUTO (Index: 6000)	61
WARM-DOWN (Index: 7000)	62
STOPPING (Index: 8000)	63

STOPPED (Index: 9000).....	64
OPERATOR ATTENTION (Index: 9800)	65
SYSTEM FAULT (Index: 9900)	66
EMERGENCY STOP (Index: 9999)	67
Outputs Section	69
Programming Add-On (AOI) Sequence's	70
System (SYSxxxx) Module(s).....	70
Indicator Push Buttons Module(s).....	71
Indicator Stack Module(s)	74
Common Programming Methods	77
Stage Triggering Rung	78
Stage String Output Rung	79
Shutdown Trigger(s) Rung	80
Latched Output Control Rung(s)	81
Non-Latched Output Control Rung(s).....	82
Warm-Up / Warm-Down Trigger Rung	83
Next Stage Trigger Rung.....	84
HMI PROGRAMMING	85
Screen Saver Screen.....	86
Main Menu Screen.....	87
Assembly Control Screen	88
System Control Screen.....	90
Sub System Control Screen	92
Installing Hardware.....	94
Indicator Stack Configuration.....	94
Assembly Stack Red Indicator	95
Assembly Stack Orange Indicator	96
Assembly Stack Blue Indicator	97
Assembly Stack Green Indicator.....	98
Operator Control Panel.....	99
Assembly / System Start Button / Indicator	100
Assembly / System Stop Button / Indicator	101
Assembly / System Reset Button / Indicator	102

Main Control Panel External Layout	103
Power & Voltage Pathways	104
Component Layout.....	105
Connection Configuration(s)	107
Documentation Overview	109
Schematic Legend / Title	109
System Overview Page	109
Programming Overview Page.....	110
Summary of Recommended Design Principles	111

Licensing Agreement

Copyright (C) Phillip Kraguljac. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".



Introduction

The following document is a framework for programmers and machine asset owners. This is not an exhaustive list as to how machines are to be programmed but a overarching guide to align programmers' programming architectures with owners' expectations. There may be variations between this framework and equipment programming as a result due to specific equipment requirements.

This guide has been developed to provide a common standard for programming as well provide asset owners a better understanding of how their equipment works. It provides asset owners with a standard of programming which can be implemented across their machinery. By standardising machine programming across multiple machines – it should assist with reducing equipment downtime during breakdowns and reduce the cost for system upgrading.

IMPORTANT: This guide is not to be used in place of a certified safety system. This is only for the purpose of functional control of an industrial machine. Safety systems require accurate risk assessments and comprehensive control measures – which this guide does not cover or include. It is also important that this guide is not used for control systems that are exposed to hazardous environments.

Warning

This website was NOT developed for replacing official training material and or maintenance instructions. It has been developed, for electrical professionals and enthusiasts, to align their projects with the uniform methodology of programming and component layouts. This website is used as a supplemental reference for competent people within their field of training. Although reasonable care has been taken to provide accurate information, this site assumes no responsibility for any consequence arising out of the use of this presentation.

There may be requirements by local laws and legislation that appropriate licensing and registrations may need to be held before installing and or modifying electrical systems. Please contact your local electrical and telecommunication authorities for more information.

IMPORTANT: Remember all sources of electrical power must be disconnected prior to visual inspection, testing, maintenance and or installation of equipment - regardless of voltage and or current levels.

Benefits of Open Sourcing

This document is released under an open-source licensing model to promote transparency, accessibility, and long-term sustainability of control system design. Open sourcing enables clear visibility for all stakeholders, improves cross-compatibility between systems, and allows asset owners to engage multiple service providers without incurring additional costs associated with proprietary restrictions or reverse engineering. By removing barriers to understanding and modification, this approach supports informed decision-making and protects long-term asset value.

Open-source design encourages community involvement and collaboration across engineering, operations, and maintenance disciplines. Shared knowledge and collective refinement lead to higher-quality outcomes, improved reliability, and more resilient systems. Ultimately, this approach shifts the focus from ownership of implementation to ownership of understanding—benefiting asset owners, integrators, and operators alike.

Additional Points

Any text highlighted in **yellow** is intended to be added directly into the PLC program as a comment, providing contextual explanation and improving code readability for engineers, technicians, and maintenance personnel.

Definitions

Add On Instructions (AOIs): A program (or function block) which is reused within the code multiple times.

Assembly: A set of processes (Systems) which when working together modifies a raw product and transforms that product into a different form.

Inputs / Outputs (IOs): These are the inputs and outputs typically connected to the interfacing slots on the PLC. Although it is not just restricted to these and can include those inputs and outputs connected to remote racks and also those devices connected via a more advanced communication medium (such as ethernet / Profibus etc).

Original Equipment Manufacturer: This is referring to the equipment manufacturer that design and manufactures the component being discussed.

Programmable Logic Controllers (PLC): This is referring to the computer that is designed to run a custom program (as discussed in this framework) for the industrial Assembly.

Product: A material which when supplied to the subject machine (Assembly) changes from one form to another form as intended by the original design of the machine.

System: A set of components (also known as Sub-Systems) which when working together influence the outcome of a product being processed through a piece of machinery (also known as the Assembly).

Sub System: A physical component which either provides an input or an output to a control system.

Equipment Overview

Machines are present in almost every premise these days. Some examples of machinery include; Fabrication Equipment (such as saws, drill presses, sheet metal presses), Bottling Equipment, Spraying Systems, Welding Systems, Crushing Units, Mixing Systems, and so forth. All these systems typically have a control system installed. Some of these control systems are managed by basic electrical circuits while others are controlled by Programmable Logic Controllers (PLCs).

Regardless of the type of control system used, they all employ the following principles. That is, the different levels of control with how it manipulates the materials (also known as products) in which it was designed for.

Argument for Adopting Modular Sequence-Based PLC Programming (MSBP)

In the world of industrial automation, control systems are the backbone of manufacturing operations. As programming engineers and engineering managers, the systems you design, implement, and maintain must be reliable, scalable, and easily adaptable to both operational changes and unexpected breakdowns. The PLC programming method outlined in this framework offers a structured, modular approach that aligns perfectly with these core objectives. Here's why adopting this method will enhance your team's efficiency, reduce costs, and ensure the long-term success of your control systems.

1. Enhanced Fault-Finding and Troubleshooting Efficiency

For engineers, the ability to quickly isolate and address faults is crucial to minimizing downtime and maintaining operational continuity. The method organizes the system into clearly defined Sub Systems and Systems, which directly translates to faster troubleshooting. When a fault occurs, the problem can be traced back to a specific System or Sub System, dramatically reducing the time spent diagnosing issues. This modular approach enables engineers to quickly locate the source of the problem without sifting through monolithic code or sprawling control systems.

Example: In a bottling system, if there's a failure in the liquid level monitoring, isolating the issue to the liquid sensor (a Sub System) instead of the entire filling process simplifies the repair process.

2. Scalability and Future-Proofing

Engineering teams are often faced with evolving operational needs or the introduction of new technologies. The structured approach outlined in this framework allows for scalable and future-proof programming. Each System is modular and can be expanded or upgraded without disrupting the rest of the production line. The simplicity of basic I/Os over more complex communication protocols (like Ethernet) ensures that new components can be integrated with minimal cost and effort, while also avoiding future compatibility issues.

Example: Adding a new sensor or actuator to an existing system can be done with minimal reprogramming, saving time on retrofitting and reducing the chance of system incompatibility.

3. Cost Reduction and Control System Optimization

For engineering management, the balance between performance and cost is always a top priority. The simplicity of using digital I/Os and analogue I/Os reduces the complexity and cost of communication between components. With fewer dependencies on advanced protocols, maintenance and upgrades become less expensive. Furthermore, the method's flexibility allows for the use of off-the-shelf components instead of being locked into proprietary technologies from specific manufacturers.

Example: When replacing outdated equipment, the method allows for easy integration of non-OEM components, reducing the overall cost of equipment replacement and system upgrades.

4. Improved Safety, Reliability, and System Control

Engineering teams need to ensure that their systems operate safely and reliably. The inclusion of Stages within each System (e.g., Standby, Starting, Running, Stopping) creates a logical flow of operations, which mitigates the risk of dangerous malfunctions or system failures. The ability to define specific actions and checks at each stage enhances system control, reduces the chance of human error, and improves overall reliability. By following a clear progression of stages, your team can prevent the system from moving into unsafe conditions or operating beyond its designed limits.

Example: The inclusion of an Emergency Stop stage ensures that the system can return to a safe state in case of a fault, without relying on separate safety systems to handle control functions.

5. Modular Programming for Faster System Updates and Maintenance

Programming engineers often deal with the need to update systems, either for performance enhancements or to accommodate new operational requirements. The modular nature of this approach—where each System resides in its own independent program module—makes upgrades and modifications easier and faster. Engineers can add or adjust functionality in one area without the need to reprogram the entire system. This not only reduces programming time but also enhances the system's overall maintainability.

Example: If a new feature needs to be added, such as an additional sensor or a different mode of operation (e.g., switching from Manual to Auto), the modification is

isolated to the specific System or Sequence involved, without affecting the broader system architecture.

6. Streamlined Operator and Technician Training

For engineering management, having a consistent, standardized approach to PLC programming is essential for training new operators and maintenance personnel. The use of Sequences and Stages provides a common language across all systems, making it easier for operators and technicians to understand machine behavior, troubleshoot, and make adjustments as needed. This consistency reduces training time and ensures that your team can handle a variety of systems with confidence and competence.

Example: A technician familiar with the Standby to Running (Auto) sequence can easily work across multiple machines without needing to relearn different control systems, streamlining the training process.

7. Reduced Risk of Programming Errors and System Failures

One of the most critical factors for programming engineers is minimizing the risk of errors during the development and modification of control systems. This method's structured approach, with its clearly defined Stages and Sequences, ensures that all scenarios are considered and handled before deployment. By establishing a set of predefined actions at each stage of operation, engineers can anticipate potential issues and program the system to handle them proactively. This reduces the risk of human error and unintended system behavior, making the system more reliable.

Example: If the Fill Line Pressure System detects that the pressure falls below a set point, the sequence automatically triggers the pump to activate. The system's logic flow ensures that these steps are followed in a safe, predictable manner.

Conclusion: A More Efficient and Reliable PLC Programming Method

For programming engineers and engineering management, adopting this structured PLC programming method isn't just a matter of convenience—it's a strategic decision that drives operational efficiency, cost savings, and safety. By organizing control logic into modular Sub Systems, Systems, and Sequences, this approach makes it easier to troubleshoot, upgrade, and maintain complex industrial machinery. It provides engineers with the flexibility to adapt to future needs, reduces the risk of

downtime, and ensures that all systems operate safely and predictably. For those looking to build scalable, maintainable, and reliable automation systems, this framework is the ideal solution.

Safety Instrumented Systems (SIS) – Scope and Exclusions

Safety Instrumented Systems (SIS) are dedicated protection systems designed to automatically place a process or plant into a safe state when predefined hazardous conditions are detected. Unlike standard control systems, SIS are required to meet strict functional safety standards and provide a measurable level of risk reduction. These systems must be designed, implemented, verified, and maintained in accordance with the applicable safety standards for the region in which they are deployed.

In Australia, SIS design and implementation are typically governed by the following standards (as applicable to the industry and application):

- **AS IEC 61508** – Functional safety of electrical, electronic, and programmable electronic safety-related systems
- **AS IEC 61511** – Functional safety – Safety instrumented systems for the process industry sector
- **AS 4024** – Safety of machinery (where machinery-related safety functions apply)
- **AS/NZS 3000** – Electrical installations (Wiring Rules), where relevant to installation practices

These standards define requirements across the full SIS lifecycle, including hazard and risk assessment, Safety Integrity Level (SIL) determination, system architecture, independence, verification, validation, testing, management of change, and ongoing maintenance.

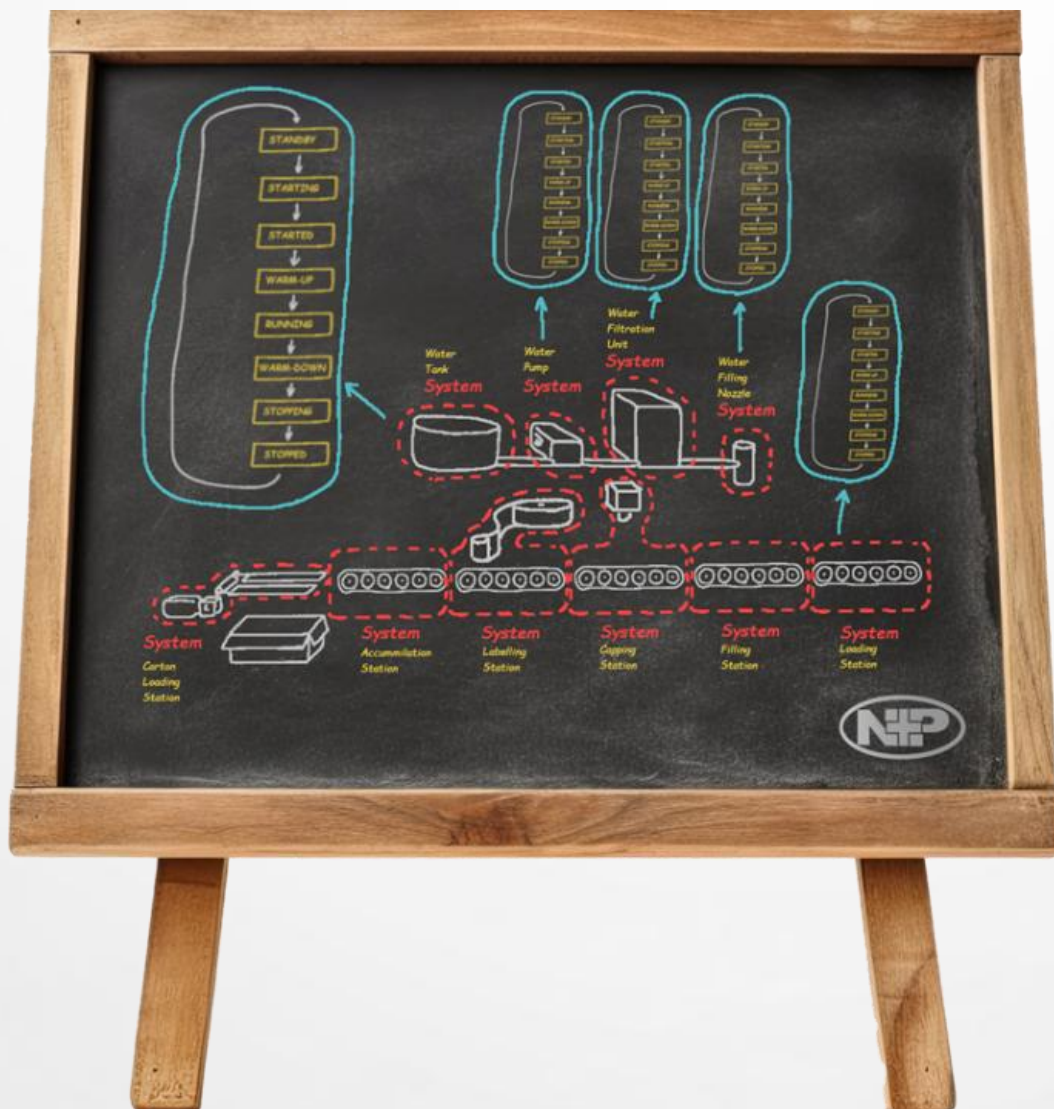
A fundamental requirement of any SIS is **independence** from the Basic Process Control System (BPCS). This includes independence in hardware, software, communications, power supply, and where required, physical segregation. SIS logic must not rely on standard PLC control code, HMI systems, or non-safety-rated components to perform its safety function. Additionally, SIS must be designed to fail safely, respond when required, and provide a defined and auditable level of certainty that the safety function will operate on demand.

It is important to explicitly state that **this document does not form any part of a Safety Instrumented System**. The design philosophies, programming structures, layouts, and recommendations contained herein apply solely to **standard control systems** and are intended to support clarity, maintainability, and operational consistency. Any references to safety-related signals (such as Emergency Stops, trips, or shutdown indications) are for monitoring, indication, or coordinated control response only and must not be interpreted as fulfilling a safety function.

All safety systems must be independently risk assessed, designed, and certified in accordance with the relevant standards by suitably qualified personnel. This document does not provide guidance on SIS design, SIL determination, safety validation, or certification and must not be used as a substitute for a compliant safety lifecycle process. Its intent is to coexist alongside properly designed safety systems, ensuring clear separation between control functionality and safety integrity.

How It Works

This framework can be used across various types of systems. It's designed in a way that it can be scaled, it is flexible for future expansion and upgrading, assists with system integration, and should assist with breakdown recovery times. What this guide does not do, is teach how to program a PLC. It is assumed that a suitable level of programming knowledge is held by the programmer.



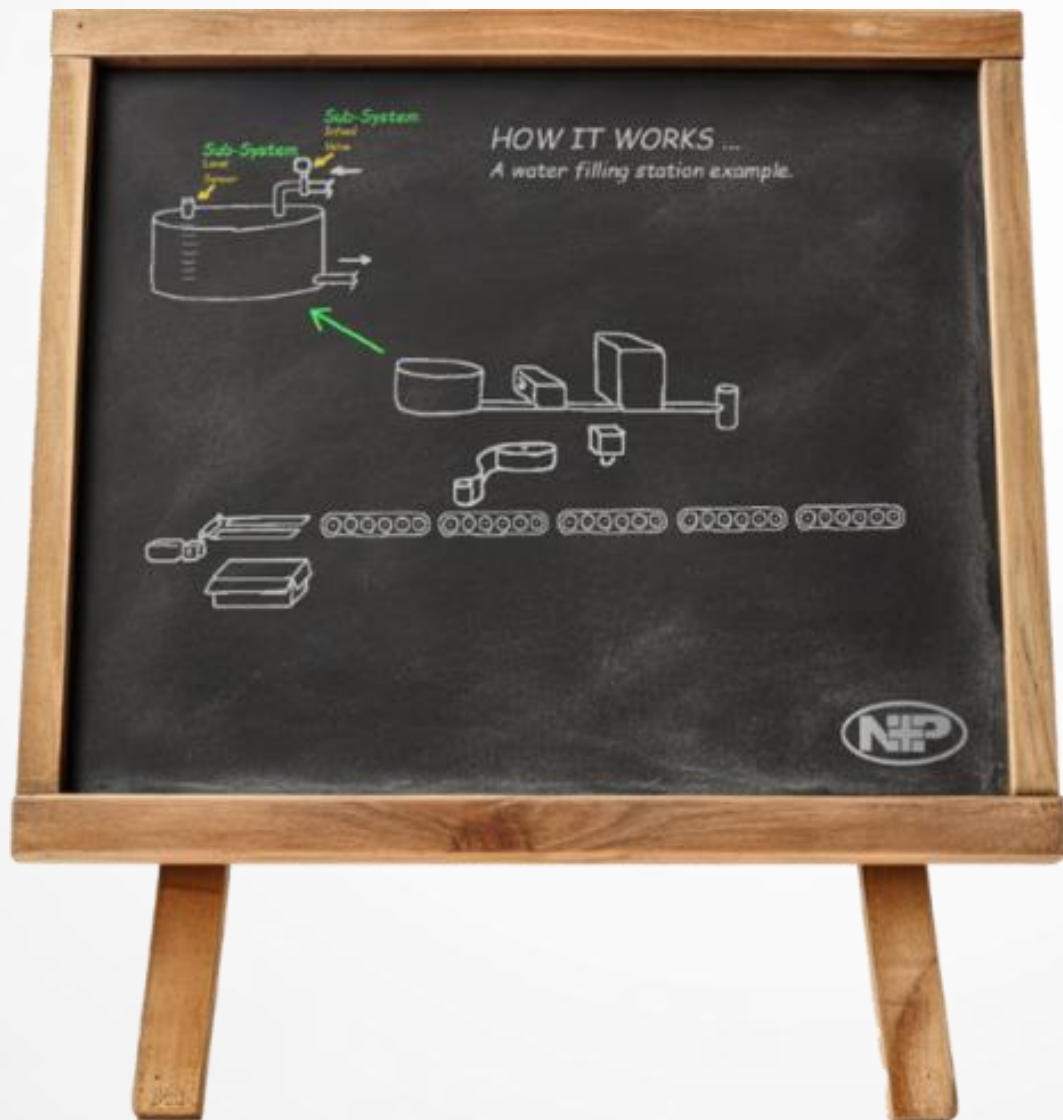
What are Sub Systems?

The best starting point in understanding how machinery works, is looking at the basic components. A machine is made up of various sensors, buttons, actuators and valves. Each of these components (for example a sensor), in the confines of this document, are classified as a Sub System. These units can be as basic as a dry contact switch or as complicated as a VSD having its own internal CPU.

These components can be connected to a more advanced communication medium such as Ethernet, however employing such technologies will increase system costs as well as reduce 'fix' options during breakdowns. There are also risks associated with differences in communication protocol versions which can result in cost increases during breakdowns when equipment has become obsolete. When keeping communication to basic forms like digital IOs and analogue IOs, this risk can be significantly reduced.

Within this document we will use the example of a bottling assembly. An example of a Sub System within this scenario, is that of a product liquid level sensor (input) which is monitoring the tanks product liquid level which is ultimately going to be used for filling bottles on the production line. Another example is that of the valve (output) which controls the flow of the fluid being supplied to that said tank. Both components are independent from one another and their only link is through the control System that either listens to their signals or tells them when to operate.

An important point to take from this is that the communications between the component should be reduced to a small number of Input / Outputs (IOs). Ideally, the number of connections needs to be kept to a minimum between these units. The reason being that it reduces the workload required for finding non 'like for like' parts during breakdown and provide greater options for what can be later used if there is a desire to change the unit from the Original Equipment Manufacturer (OEM).



What are Systems?

The next level up from Sub Systems are Systems. System is the term used to describe a relationship between various components (also known as Sub System). This is typically a single point of transformation of a product being used in the manufacturing process. It usually only involves a single step. It usually involves monitoring and controlling various Sub Systems.

An example of a System, relating to the bottling Assembly scenario, is that of the product liquid storage tank. The tank level needs to be constantly monitored and filled when nearing the empty point. The purpose of the tank is to act as a buffer between the slow product liquid supply and a high-volume discharge during the bottling filling phase. The connection of the liquid level sensor with the control valve that re-fills the tank, is what we refer to as a System. It is the system of filling the tank. Note that it does not include the draining, that would be another System.

Another example of a system could be the detection of a bottle being present within the filling location and the valve filling the liquid into the bottle. Note that the System is kept to a minimum number of interactions between Sub-Systems. This means that the System which is responsible for filling the bottles does not include moving the bottle in and out of the filling location.

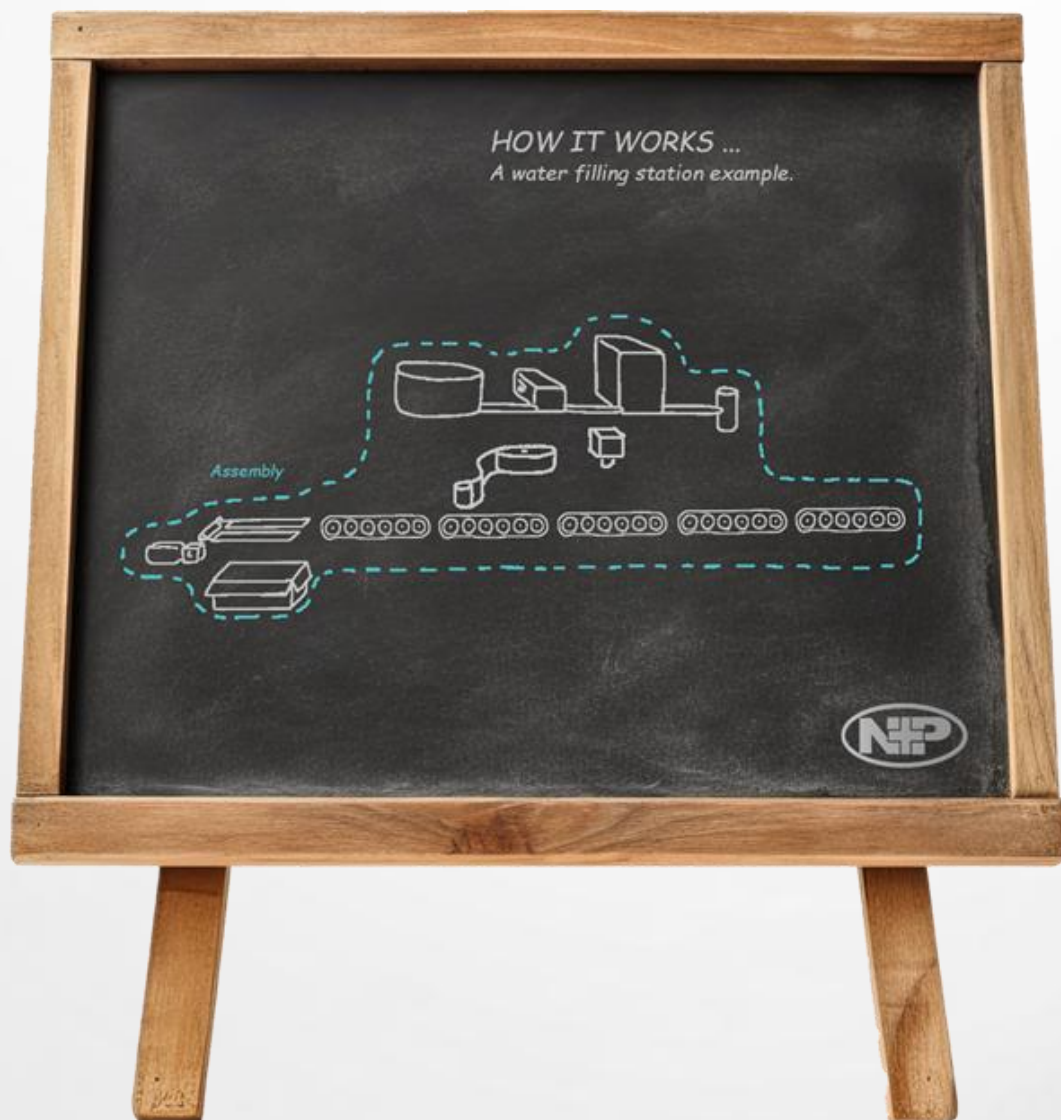
Another example of a System could be the maintenance of pressure within the product liquid fill line. The Sub-Systems utilised within this System are the product liquid fill line pressure sensor and the VSD for the motor connected to the product liquid fill line's pump. The interaction between these sub-systems (i.e. when the pressure drops below a certain amount - the pump starting) is what is known as the System. The System is maintaining the set pressure of the product liquid fill line.

Ideally, it should be possible to list all the Systems utilised within a PLC as well as having each System residing in its own independent program module. This allows for faster fault finding as well as faster and more cost-effective upgrades.

What are Assemblies?

Finally, Assemblies are the collection of Systems which ultimately transfer a product from its raw form to a more finished product.

An example of an Assembly is the filling, labelling, and capping of water bottles. The Assembly has various Systems (tank storage level management, bottle filling, bottle capping, conveyor movement, bottle labelling etc).



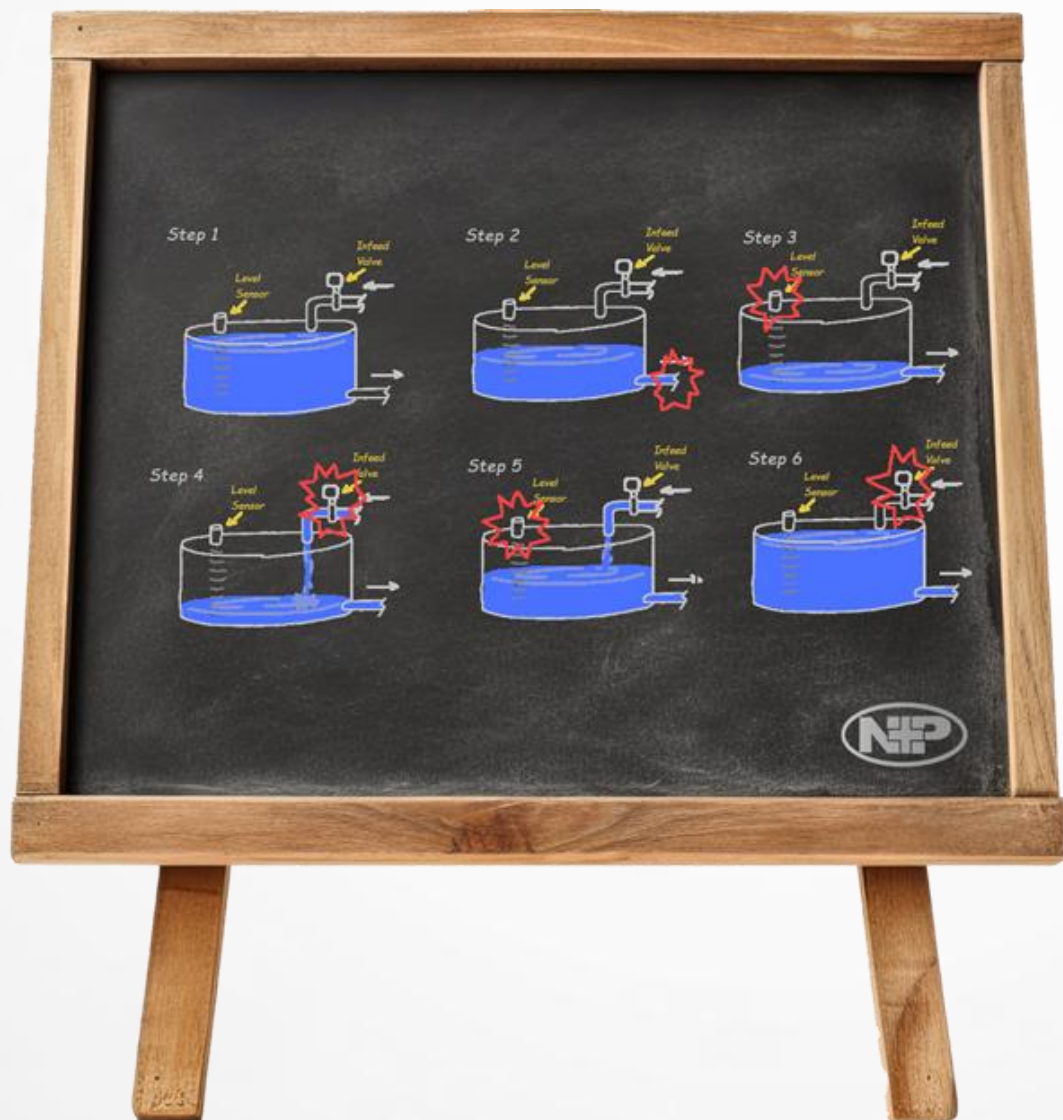
What are Sequences?

Sequences are essentially a storyline or pathway of how a machine is to operate. They are utilised in both System level programming as well as Assembly level programming^[1]. It includes both the management of the outputs (controlling valves and VSDs) as well as monitoring of inputs (buttons and sensor readings). It is the logical progression of events during a machine's operation.

An example of a sequence is the product tank liquid level monitoring System. Initially, when the tank is full, the System (filling the tank) is in the **Standby Stage**. Once the water level drops below a set low level point, the Sequence then goes into a **Filling (Run) Stage**. It then subsequently triggers the inlet valve to open. Once the product liquid level climbs to the high point, the Sequence goes into a **Stop Stage** which then triggers the inlet valve to close. Once the Sequence detects that the valve has closed, and the level is no longer increasing then the Sequence switches to the **Standby Stage** again. This process is then repeated as often as required.

Sequence programming is typically included within the System and Assembly programming modules.

^[1] Not to be confused with the low-level programming that is transferred to the PLC. For the purposes of this framework Assembly Programming is that which involves the interconnection programming that links the system programming.



What are Stages

Stages are the sub-set of a Sequence. They are the steps in which a machine takes as opposed to the entire storyline (which would be referring to the Sequence).

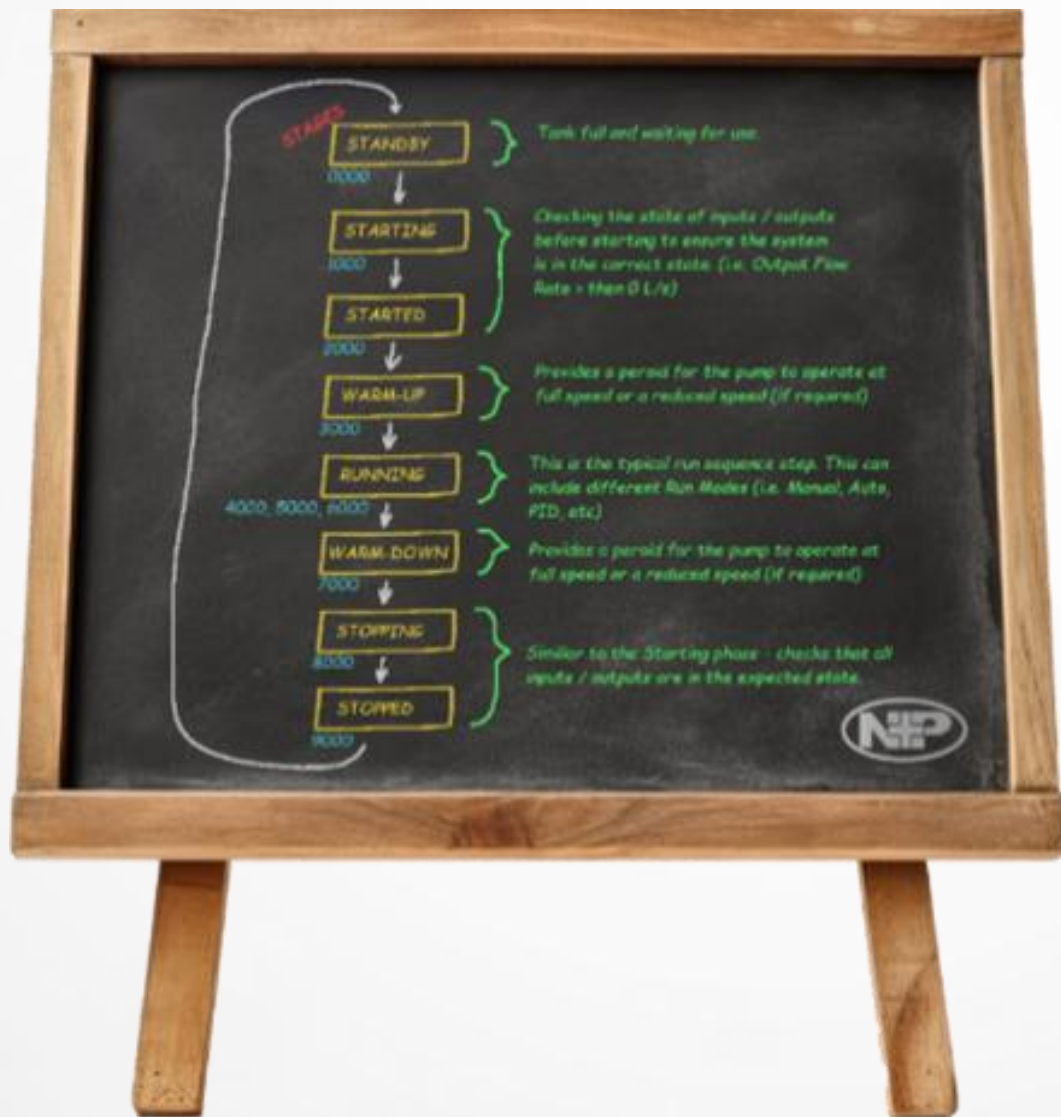
Depending on what conditions are present at the time, the machine will jump to the required Stage as pre-determined within the Sequence programming. For example; if an operator presses the Run button while the machine is in the Standby stage, the machine will progress to the Starting stage.

As part of this framework there are 10 fundamental stages;

1. Standby
2. Starting
3. Started
4. Warm-up
5. Running (Initial)
6. Running (Manual)
7. Running (Auto)
8. Warm-Down
9. Stopping
10. Stopped

Depending on the system requirements, there may be more present, however as a fundamental all of these stages should exist in every System. This is to ensure that in the chance that the System needs to be upgraded, and or improved, these items are not having to be added and integrated at a later stage into a working system. Making major modifications into a working system can increase the risk of equipment damage and malfunction during programming.

This framework allows for greater flexibility when adding stages (which is a common request during the life cycle of equipment). If implemented across multiple machines it allows for greater understanding by programmers and technicians during breakdown. It also allows for greater visibility of System status across all levels within the machine. It also lowers the risk of incorrect programming. This method of programming also allows for safer cut-overs when programming on live equipment.



Essentially Sequencing operates on an indexing variable. The corresponding values for each of the stages are detailed below.

Standby (Index: 0000 → 0999)

Within this Stage the System is not operating and is waiting to be initiated. This can either be manually triggered via operator intervention or via an AUTO Run Switch / Latch. Once a Sequence has finished its Stopped Stage, it should return back to this Stage. During this Stage, no output should be operating.

Starting (Index: 1000 → 1999)

This Stage typically involves the System conducting pre-start checks before running. These can include making sure consumable levels are correct, and or doors are closed etc. This is different from fault interrupters which occur during the system Run Stages. These checks are usually those System readings which are present while the system is operating.

An example of this would be the System checking the pressure on the product liquid fill line. As the system would not be running, the expected pressure reading should be near zero. If the System detects a high-pressure reading, then the System could conclude that there is a fault which maintenance crews need to investigate.

This Stage can also include other functions like opening supply lines to the System.

Started (Index: 2000 → 2999)

This Stage is essentially the point in which the above checks have passed, and the System can then progress to the next Stage. This Stage can also include checks that the actions taken in the previous stage have successfully executed (for example checking those pressures in the supply line (downstream of the valves are within an expected range).

Warm-up (Index: 3000 → 3999)

This Stage is typically used for warming equipment up (if required). It can be used for charging lines as well as for testing liquid lines at lower pressures prior to engaging full pressure (which would be present during normal operation). Thereby reducing the amount of spillage in the event of pipe rupture. Having the system start at a lower pressure may also reduce the effects of hydraulic shock. When using this stage, a timer will be needed, which switches over to the Running stage (4000) after a pre-determine amount of time. If this Stage is not required, then program the system to immediately jump to the next Stage (4000).

Running [Initial] (Index: 4000 → 4999)

This Stage is the initial Stage of Running. Within this Stage the System typically determines which Running mode is required. It can do this through checking user inputs and or equipment states. An example of this is if the user has selected AUTO on the mode switch on the control board. As part of this Stage, the System would check the state of that switch and then change the Stage to Running (Auto) (6000).

Typically, when a Stage is switched out (for example being switch out of AUTO to Manual mode by an operator), the Sequence will then be switched back to this Stage (4000) which will then determine that the next stage will be Run Manual mode (5000).

Running [Manual] (Index: 5000 → 5999)

In this Stage the System / Assembly is being manually ran. System setpoints are typically defined by the operator / maintenance staff. When the System / Assembly detects a change in Run Mode (for example Manual → Auto) then the Stage will be changed to Running (Initial) (4000).

Running [Auto] (Index: 6000 → 6999)

This Stage is the typical stage that's going to be employed most of the time. For complex Systems which may have several steps, additional Stages can be added (for example 6001, 6002, 6003 and so forth).

An example of this could be a product liquid fill line pressurisation system which operates at two pressures (Standby and Fill). When the System isn't filling bottles with liquid, the Stage is set to Running Auto Standby (6000). When the System detects that it is about to fill a bottle (via monitoring another System's status which is responsible for filling product into the bottles) then it sets its own Sequence Stage to Running Auto Fill (6001). When the System responsible for Bottle Filling changes to completed, the Pressure System sets its own Sequence back to the Running Auto Standby (6000) stage.

Warm-Down (Index: 7000 → 7999)

Please refer to Stage Warm-up (3000) as this is the reverse function of that Stage.

Stopping (Index: 8000 → 8999)

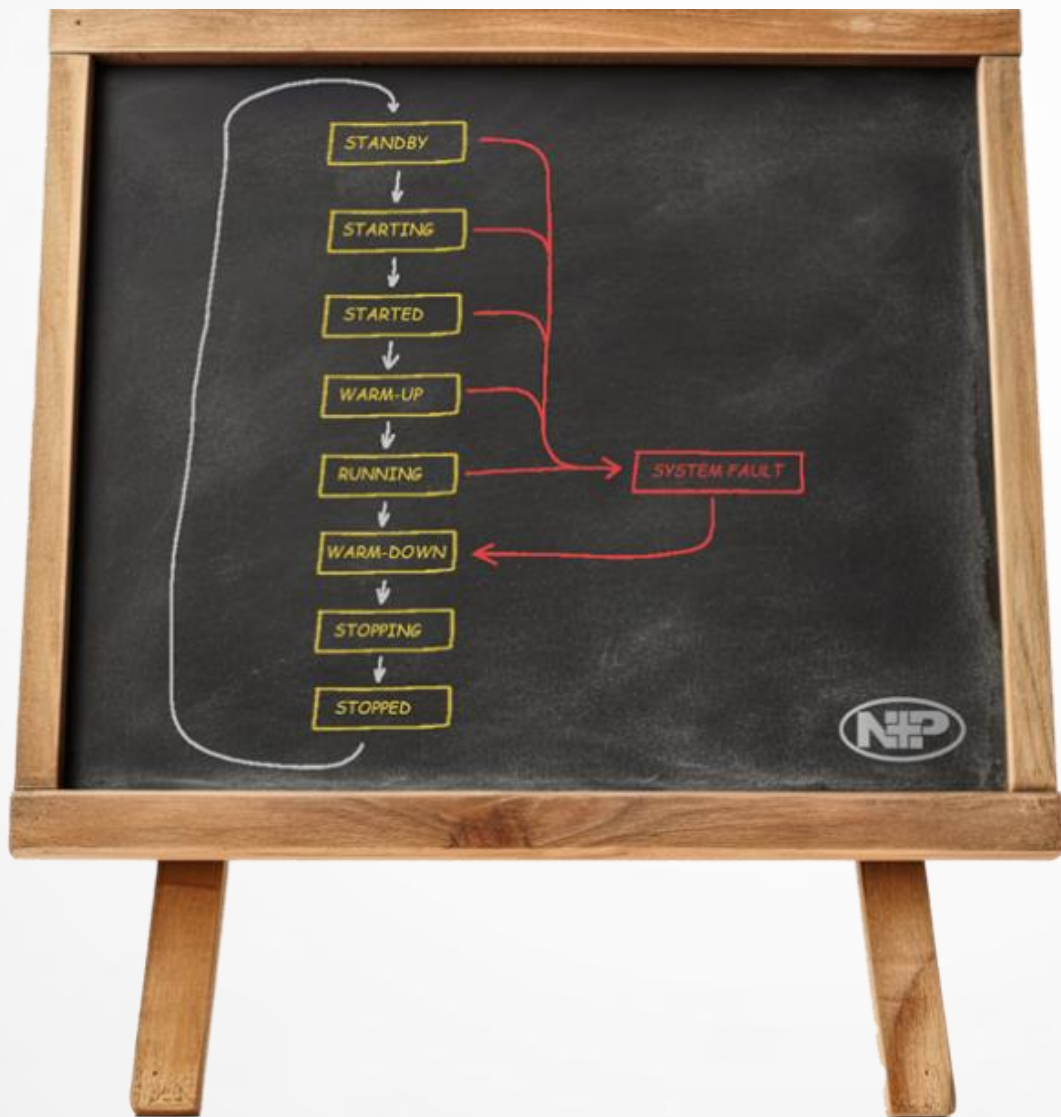
Please refer to Stage Starting (1000) as this is the reverse function of that Stage.

Stopped (Index: 9000 → 9499)

Please refer to Stage Started (2000) as this is the reverse function of that Stage.

Operator Intention (Index: 9500 → 9899)**System Fault (Index: 9900 → 9998)**

This Stage is for setting the system into a state which is acceptable under any fault conditions. The purpose for employing a Fault Stage is to ensure that when a fault does occur, the System is then deliberately interrupted, and requires a deliberate action by the operator and or maintenance to acknowledge the fact that an error has occurred.

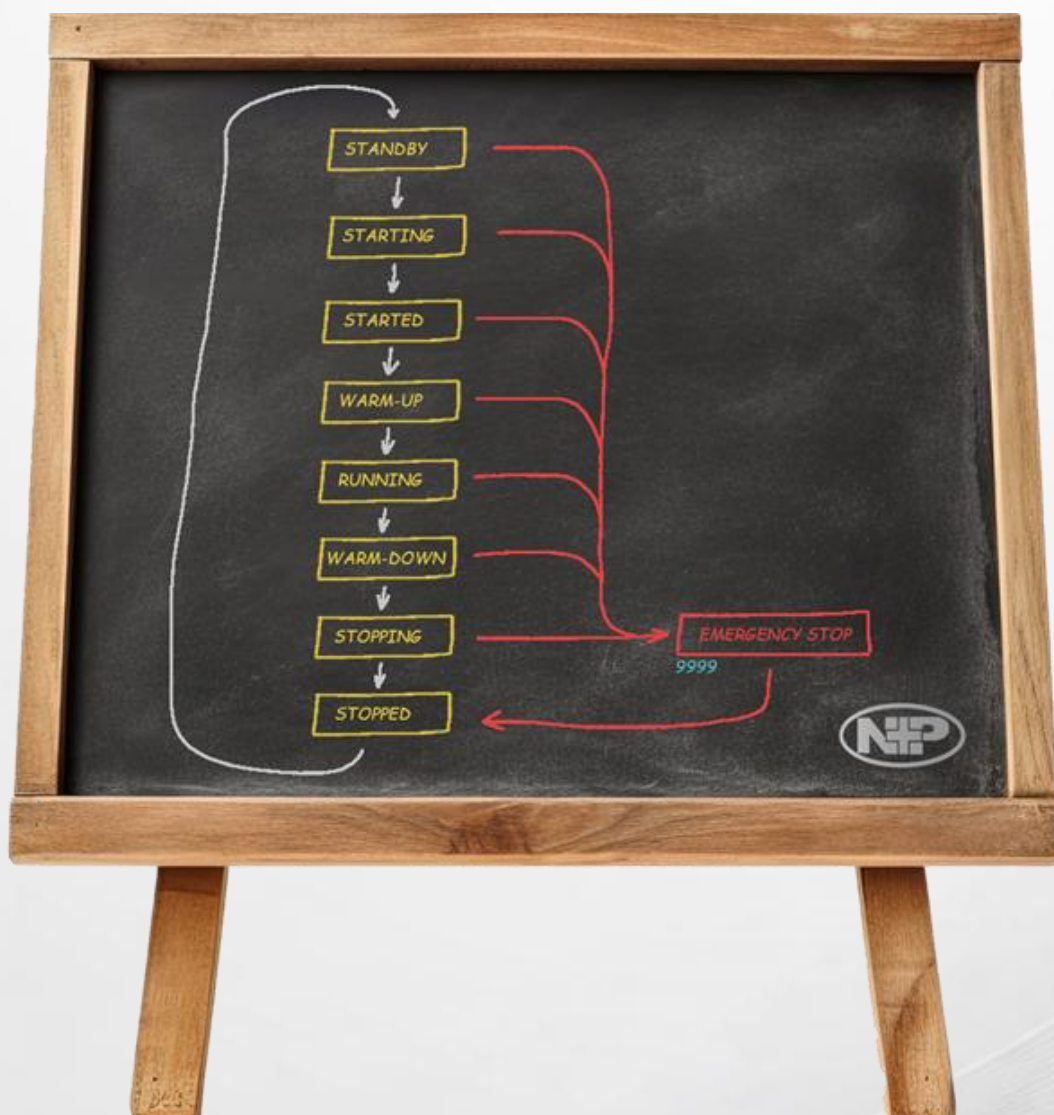


Emergency Stop (Index: 9999)

IMPORTANT: This Stage is not a replacement for having a certified safety system. This is a supplementary and informative function to allow the system to return to a state ready for resetting.

The emergency stop Stage is purely for stopping the control system outputs and ensuring that they ready for resetting. This stopping function is purely supplementary to the emergency control system and is only there if the safety system fails, the control system isn't forcing the outputs to trigger. It is also, used for setting System / Assembly into a state ready for it to be return into service.

In the same manor that the Fault Stage works, the system requires acknowledgement from the operator and or maintenance staff that the system has been placed into emergency mode. Within this section, any emergency indications which are added, must not be latching in nature.



Programming Main Sequence & System Sequence(s)

Modification Notation

It is recommended, that any modifications made to any of the programming should be clearly recorded and commented within the project. This allows for programmers to quickly revert back to known working code as well as contact any programmers regarding any questions they may have regarding the code that has been added. It also allows asset owners to contact the Service Providers if any future changes are required.

[YYYYMMDD] [Initial].[Last Name] [Company] [Description of Change]

Input Section

The following section is for mapping inputs into working variables. This assists with system upgrades, by reducing the number of points which need to be modified when a PLC is changed. It also helps when upgrading a system by allowing the programmer to easily switch the output from existing code to the new code. If inputs are not programmed in a way which allows for greater flexibility, it can present serious issues for programmers conducting live changes to existing and running systems.

It is also recommended that the inputs which are directly associated with that programming (module), be retained within the same module, unless it is used in others. At which point the input should be programmed into a common mapping module. The reason behind this is, if a System is going to be worked on, all the associated inputs are within the same logical area. This helps with any work being conducted post commissioning by programmers, who are unfamiliar with the system.

==== INPUTS ====

SECTION DESCRIPTION: The following section is for programming all inputs specific for the control of this program module. Any inputs which are used in multiple modules will be listed within the common input mapping module.

Basic Digital Transfer

Mapping PLC hardware inputs and outputs to internal working variables establishes a clear separation between physical I/O and control logic, forming a stable and maintainable control architecture. In this approach, raw PLC I/O addresses are assigned once to clearly named internal variables, which are then used throughout the program for all logic, interlocks, and functions. This abstraction improves code readability, ensures consistent signal usage, and allows engineers to focus on system behaviour rather than hardware-specific addressing.

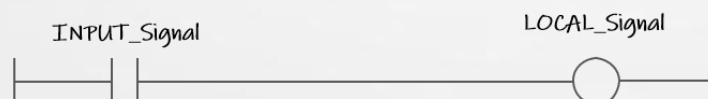
A key benefit of this method is the long-term flexibility it provides, particularly when hardware changes are required. If a PLC, I/O module, or rack configuration is replaced or reconfigured, only the I/O mapping needs to be updated—there is no requirement to search through the entire codebase to modify individual hardware references. This significantly reduces engineering time, lowers the risk of introducing errors, and enables faster upgrades or platform migrations. Overall, I/O mapping to internal variables enhances robustness, simplifies maintenance, and futureproofs the control system.

RUNG DESCRIPTION: The following rung is used to map a raw input signal into a local working variable for use within this program module. This abstraction allows the control logic to reference a standardized internal signal rather than a direct input.

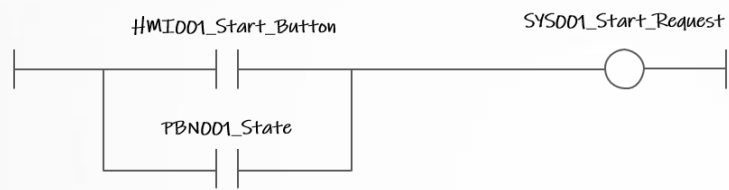
C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



(FOR MAIN LOOP ONLY)



Basic Analog Transfer

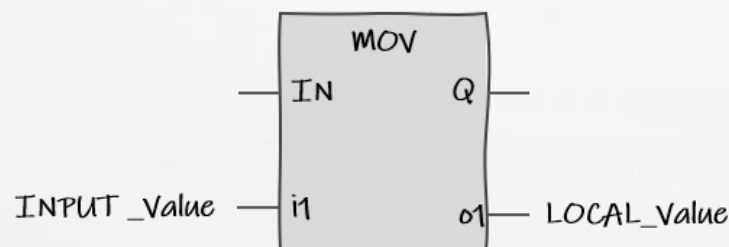
Mapping analogue PLC inputs and outputs to internal working variables separates physical hardware from control logic, allowing raw signals to be scaled, filtered, and validated in one place before being used throughout the program. This improves clarity and consistency while avoiding repeated conversion logic. If analogue hardware or the PLC platform is later replaced or reconfigured, only the I/O mapping and scaling need to be updated, eliminating the need to modify control logic and significantly reducing engineering effort, risk, and downtime over the system's lifecycle.

RUNG DESCRIPTION: The following rung is used to map a raw input signal into a local working variable for use within this program module. This abstraction allows the control logic to reference a standardized internal signal rather than a direct input.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



Scaling Inputs

Using dedicated scaling functions for analogue signals provides a consistent and reliable method of converting raw input values into meaningful engineering units before they are used by control logic. Centralising scaling in a single function ensures that limits, linearisation, fault handling, and default behaviours are applied uniformly, improving accuracy and readability while preventing duplicated or inconsistent calculations across the program.

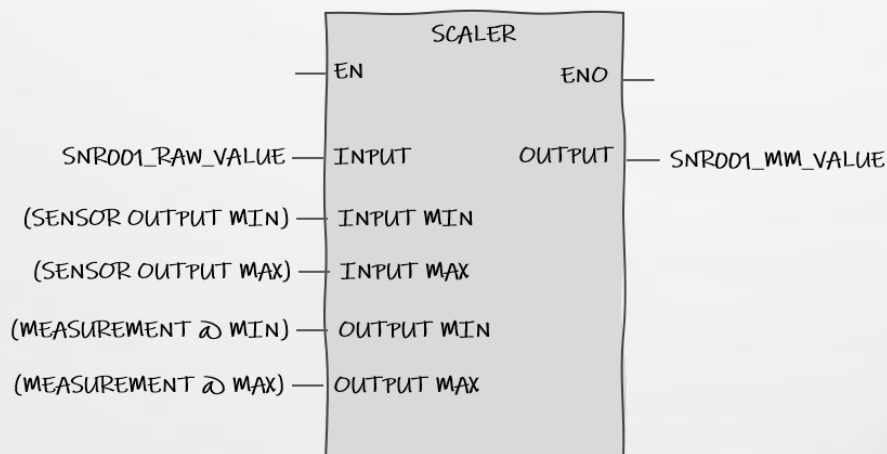
This approach also simplifies future changes and maintenance. If a sensor range, signal type, or PLC hardware is modified, only the scaling function needs to be updated rather than multiple references throughout the code. This reduces commissioning time, lowers the risk of errors, and supports faster upgrades, making the control system more robust, maintainable, and adaptable over its operational life.

RUNG DESCRIPTION: The following rung is for mapping RAW Analog inputs into SCALED Working variables.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



HMI Input Filtering

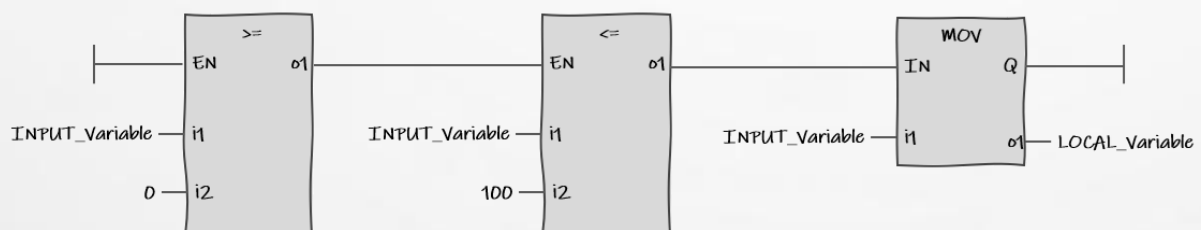
Applying centralised MIN/MAX filtering to HMI-to-PLC inputs ensures that operator-entered values remain valid and within defined safe operating limits before being used by control logic. By handling range validation at a single interface point, the system is protected from input errors and unintended behaviour while eliminating the need for repeated checks throughout the program. This simplifies future changes to operating limits, reduces engineering effort, and improves the robustness and maintainability of the HMI–PLC interface over the system lifecycle.

RUNG DESCRIPTION: The following rung is for mapping HMI user defined inputs into FILTERED Working variables.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



Default Parameter Restoration Section

The following section is for listing all critical default user defined variables. The purpose behind this code is to provide a working benchmark for programmers, if there are any doubts regarding the suitability of settings. Typically, once the System / Assembly has been commissioned, the settings, which have been proven to work, are loaded into this section. It is also possible to have the 'System Default Reset Flag' linked to an HMI which allows operators to switch back to a known working state, allowing them to self-recover if they have made an error entering a wrong value into the HMI.

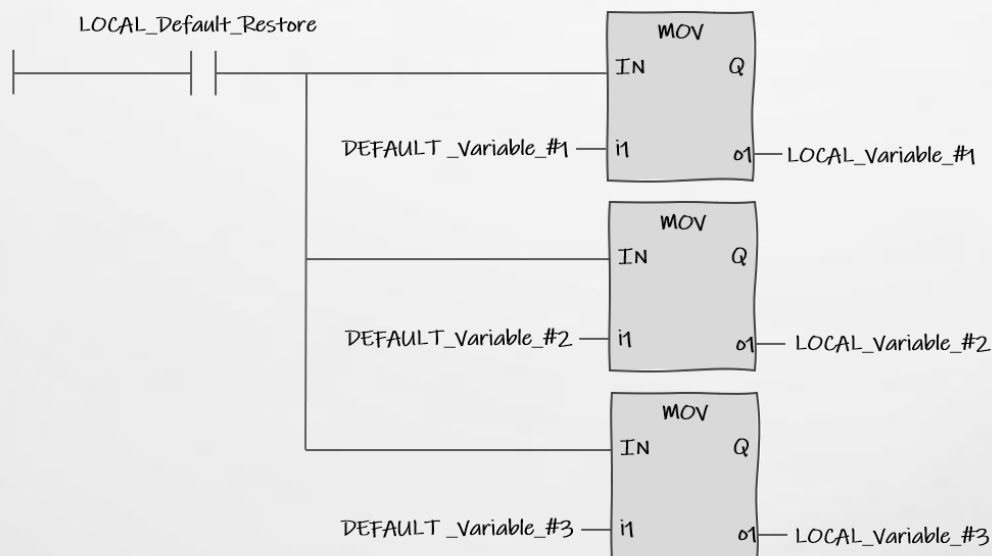
==== DEFAULT SETTINGS ====

SECTION DESCRIPTION: The following section is for resetting user defined variables back to known working values. It is highly recommended that these values are updated at the completion of final commissioning.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



General Operation Section

Reserving a dedicated section of code for **general operations** provides a clear, centralised location for functions that affect overall module or system behaviour, such as latching auto-start logic, mode management, or common interlocks. Grouping these shared functions outside of specific control sequences improves code clarity and ensures that system-wide behaviour is defined consistently and transparently, rather than being duplicated or scattered throughout the program.

This structure simplifies maintenance and future modifications by allowing changes to be made in one clearly defined area without unintended side effects elsewhere in the code. When system-level behaviour needs to be adjusted or extended, engineers can quickly locate and update the relevant logic, reducing commissioning time and lowering the risk of errors. Overall, a dedicated general operations section improves readability, robustness, and long-term maintainability of the control software.

==== GENERAL OPERATIONS ====

SECTION DESCRIPTION: The following section is for any operation which is required to be process regardless of the Sequence Stage.

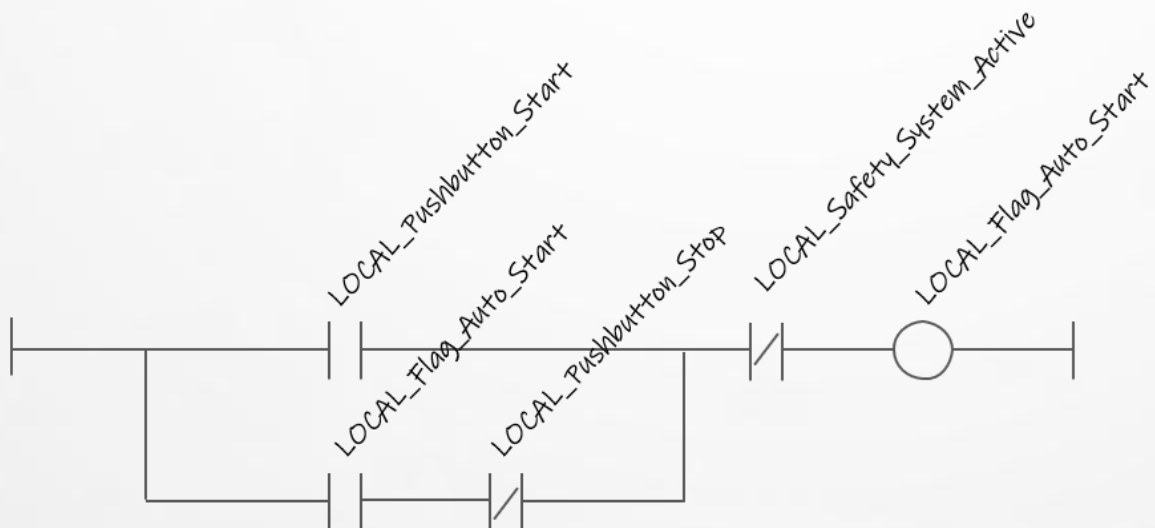
Auto Run Function

RUNG DESCRIPTION: The following rung is for providing a latching Run Auto function.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



Fault Section

Like the **General Operations** section, dedicating a specific section of code to **fault monitoring** provides a clear and consistent framework for managing system protection. This section is processed every program cycle and contains all logic required to evaluate fault conditions and set positively switched fault variables when abnormal states are detected. Centralising fault detection ensures that monitoring is continuous, predictable, and independent of individual control sequences.

By isolating fault logic in one location, sequence stages can simply reference standardised fault variables rather than duplicating detection logic throughout the program. This improves clarity, reduces complexity, and makes future fault additions or modifications straightforward. The result is a more robust, maintainable, and easily diagnosable control system, with a clear separation between fault detection and sequence response.

==== FAULT MANAGEMENT ====

SECTION DESCRIPTION: The following section is for monitoring fault condition. Any faults detected are linked to a flag variable which is then programmed into the Sequence Stage.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)

Time Monitoring

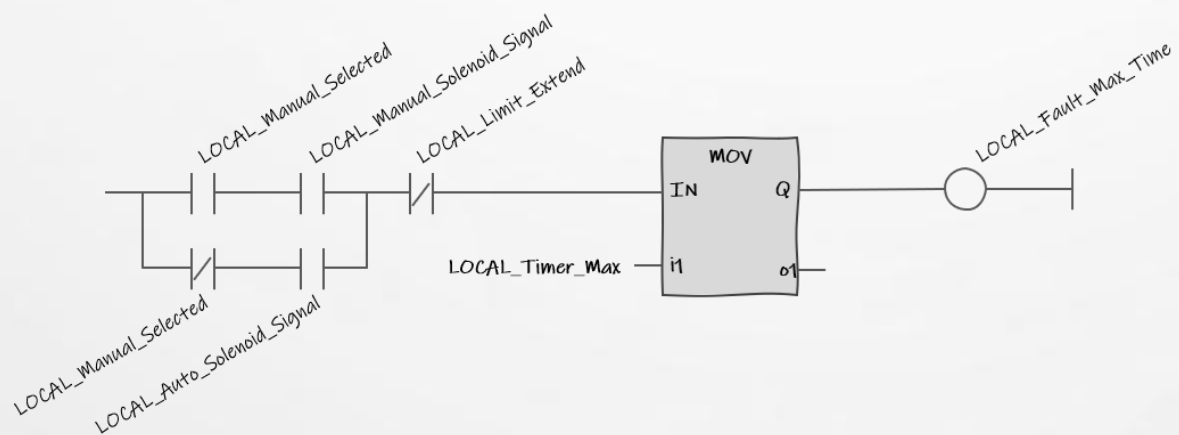
This rung monitors hydraulic valve movement by confirming that the correct limit signal is reached within a defined stroke time. A timer is initiated when valve movement is commanded, and if the corresponding limit is not achieved before the timer expires, a fault condition is generated, indicating a failure to reach the required position within the allowable time.

RUNG DESCRIPTION: The following rung monitors hydraulic valve movement by verifying that the appropriate limit signal is reached within a defined time period. If the limit is not achieved within the allowed stroke time, a fault condition is generated.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



Low Level Monitoring (with Nuisance Protection)

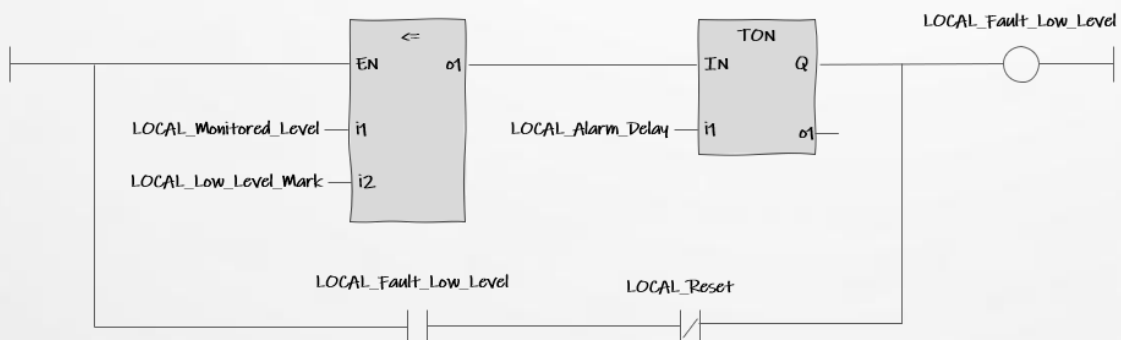
This function monitors the low level condition of a liquid storage tank to ensure adequate process availability and protection. When the low level (L) threshold is reached, the condition is detected and handled as part of normal operation; however, the low-low level (LL) is reserved for fail-safe protection in the event the primary low level is missed or not acted upon. If the LL condition is detected, a fault or protective action is initiated to place the system into a safe state, ensuring equipment protection and preventing process or mechanical damage.

RUNG DESCRIPTION: The following rung is for detecting a low value reading.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



High Level Monitoring (with Nuisance Protection)

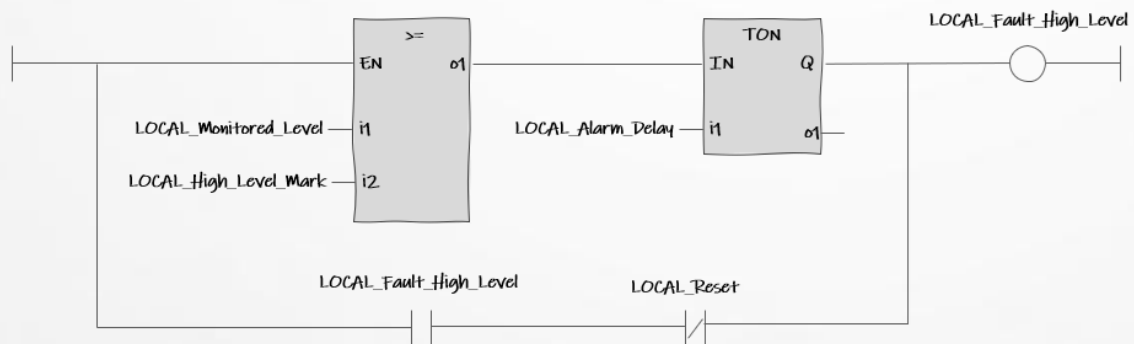
Please refer to the Low Level Monitoring section as this is the inverse of that.

RUNG DESCRIPTION: The following rung is for detecting a high value reading.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



Range Monitoring (with Nuisance Protection)

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)

[Reserved for Future Use]

Contactor Monitoring (with Nuisance Protection)

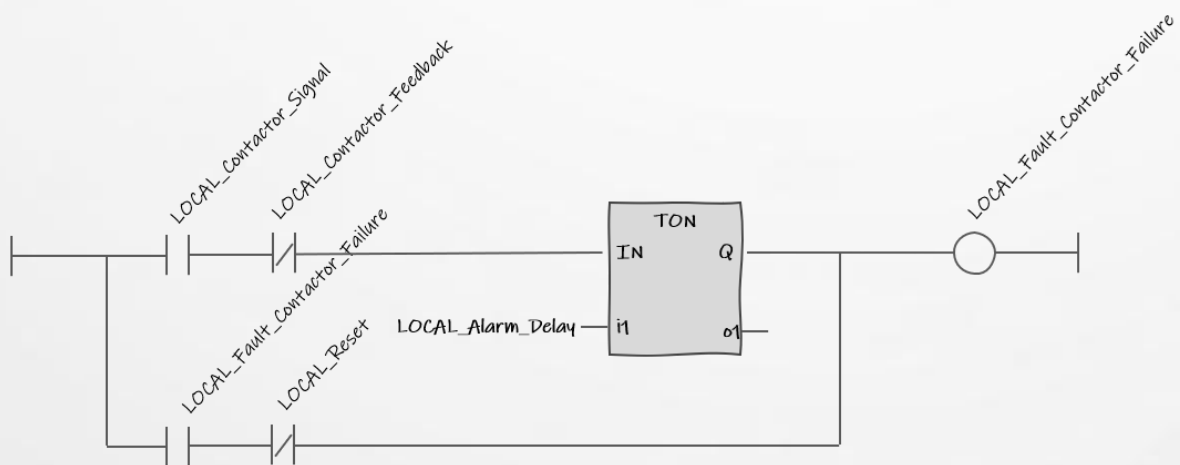
This rung monitors the status of the contactor to confirm correct operation, using a timer function as a debounce filter to eliminate transient signals or contact bounce. The contactor state is only considered valid once the signal remains stable for the defined time period, ensuring reliable detection and preventing false status changes or nuisance faults.

RUNG DESCRIPTION: The following rung is for detecting the condition of contactor.
The Timer function is included as a debounce filter.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



Fault Signalling (Immediate Shutdown)

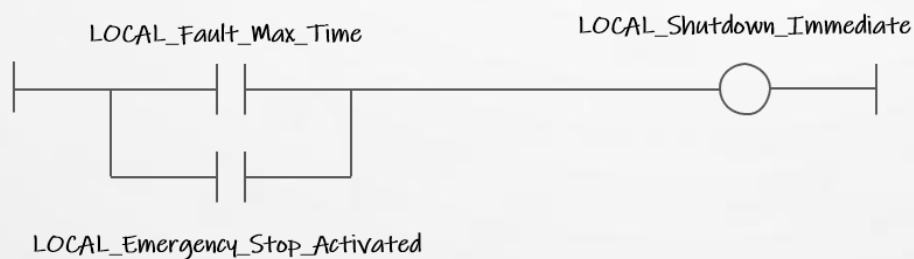
This function generates an immediate shutdown flag when a fault condition is detected, overriding normal operational logic to place the system into a safe state without delay. Once set, the shutdown flag inhibits active commands and initiates predefined protective actions to prevent equipment damage or unsafe operation. Centralising this response ensures consistent fault handling, predictable system behaviour, and rapid risk mitigation across all affected modules.

RUNG DESCRIPTION: This function generates an immediate shutdown flag when a fault is detected, overriding normal operation to place the system into a safe state without delay. Once active, it inhibits commands and triggers protective actions to ensure consistent and predictable fault response across the system.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



Fault Signalling (Controlled Shutdown)

This function initiates a controlled shutdown sequence when a fault or stop condition is detected, allowing the system to transition to a safe state in an orderly and predictable manner. Rather than immediately inhibiting all operations, the controlled shutdown manages outputs, timing, and dependencies to safely stop processes, depressurise or unload equipment where required, and prevent secondary faults. This approach protects mechanical components, maintains process integrity, and ensures consistent, recoverable system behaviour during abnormal conditions.

RUNG DESCRIPTION: This function initiates a controlled shutdown when a fault or stop condition is detected, allowing the system to transition to a safe state in an orderly and predictable manner. Outputs and timing are managed to safely stop processes and prevent secondary faults while protecting equipment and maintaining system integrity.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)

PID Section

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)

[Reserved for Future Use]

Sequence Section

As discussed previously, the sequencing section is a step-by-step process. As the system progresses through its operational phases, the PLC jumps from one Sequence Stage to another. This allows for greater flexibility with future modifications and allows for greater control when working on a live system.

STANDBY (Index: 0000)

This section places the system into an **idle state**, where no active process or motion is commanded and the system remains in a safe, ready condition. During this state, outputs are typically de-energised or held in a neutral position, ensuring that the system is stable while awaiting further commands. Defining an explicit idle state improves operational clarity, supports safe transitions between modes, and provides a known baseline condition for startup, shutdown, and fault recovery.

==== SEQUENCING [STANDBY] ====

SECTION DESCRIPTION: The following section for having the System placed in an 'idle' type state. Typically, there should be Outputs engaged during this state.

Please refer to the 'Common Programming Methods' section for the following:

- Stage Triggering Rung
- Stage String Output Rung
- Shutdown Trigger(s) Rung (only Immediate)
- Next Stage Trigger Rung

STARTING (Index: 1000)

This section performs system or assembly pre-start checks and manages the energisation of required supplies prior to normal operation. It verifies that permissives, interlocks, and initial conditions are satisfied before allowing the system to progress, ensuring that power, utilities, and supporting services are safely and correctly established. Centralising these checks improves startup reliability, prevents unsafe operation, and provides a controlled transition from idle to active system states.

==== SEQUENCING [STARTING] ====

SECTION DESCRIPTION: The following section is typically for completing System / Assembly Pre-Start Checks and or energizing supplies.

Please refer to the 'Common Programming Methods' section for the following:

- Stage Triggering Rung
- Stage String Output Rung
- Latched Output Control Rung(s)
- Shutdown Trigger(s) Rung
- Next Stage Trigger Rung

STARTED (Index: 2000)

This section performs system or assembly post-start checks to confirm that equipment and processes have reached their expected operating states after startup. It verifies feedback signals, operating conditions, and performance criteria to ensure the system is running correctly and safely. Centralising post-start validation improves fault detection, supports stable operation, and provides confidence that the system has successfully transitioned into normal running conditions.

==== SEQUENCING [STARTED] ====

SECTION DESCRIPTION: The following section is typically for completing System / Assembly Post-Start Checks.

Please refer to the 'Common Programming Methods' section for the following:

- Stage Triggering Rung
- Stage String Output Rung
- Shutdown Trigger(s) Rung
- Next Stage Trigger Rung

WARM-UP (Index: 3000)

This section manages the warm-up of equipment and the charging or conditioning of fluid lines prior to full operation. It allows temperatures, pressures, or flow conditions to be gradually established within defined limits, reducing mechanical stress and preventing shock to components. Incorporating a controlled warm-up or charging phase improves equipment longevity, process stability, and overall system reliability before transitioning to normal operation.

==== SEQUENCING [WARM-UP] ====

SECTION DESCRIPTION: The following section is for Warming-Up Equipment and or Charging Fluid Lines.

Please refer to the 'Common Programming Methods' section for the following:

- Stage Triggering Rung
- Stage String Output Rung
- Shutdown Trigger(s) Rung
- Warm-Up / Warm-Down Trigger Rung
- Next Stage Trigger Rung

RUNNING ► INITIAL (Index: 4000)

This section determines which operating mode the sequence should transition to based on current conditions and operator selection. It typically evaluates whether the system should operate in Manual or Automatic mode, though additional modes may be supported as required. Centralising mode selection logic ensures clear, controlled transitions between modes and provides predictable system behaviour, improving both operational safety and ease of use.

==== SEQUENCING [RUNNING - INITIAL] ====

SECTION DESCRIPTION: The following section is for checking which Mode the Sequence needs to be switch to. This typically falls within Manual and Auto (although not limited).

Please refer to the 'Common Programming Methods' section for the following:

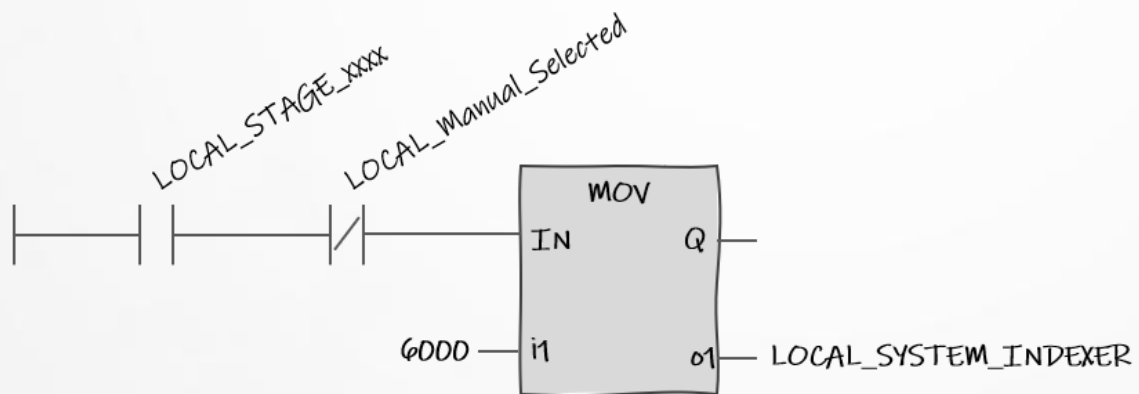
- Stage Triggering Rung
- Stage String Output Rung
- Shutdown Trigger(s) Rung
- Next Stage Trigger Rung

Automatic Mode Transition Rung

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)

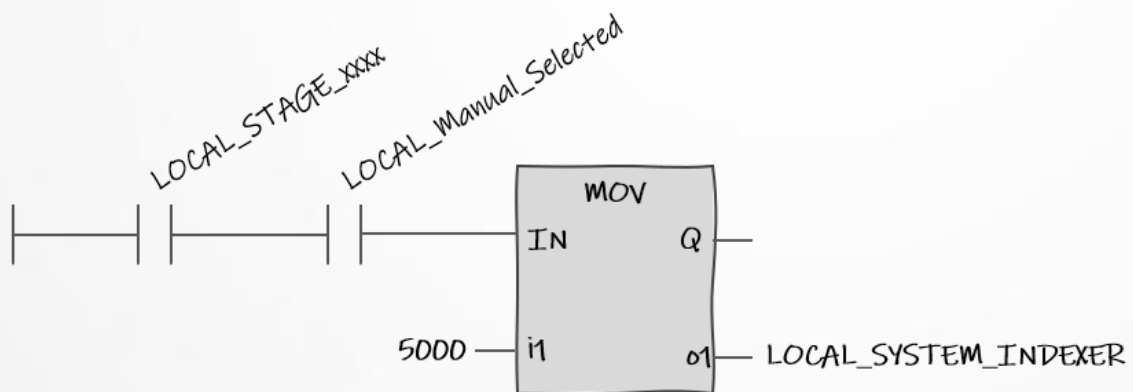


Manual Mode Transition Rung

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



RUNNING ► MANUAL (Index: 5000)

This section governs operation of the system in **Manual Mode**, allowing operators to directly control individual functions or devices while maintaining defined interlocks and safety conditions. Manual operation supports commissioning, testing, and maintenance activities by providing controlled, step-by-step interaction with the system. Isolating manual logic in a dedicated section improves clarity, reduces risk, and ensures manual actions remain predictable and safe.

==== SEQUENCING [RUNNING - MANUAL] ====

SECTION DESCRIPTION: The following section is for running the System in Manual Mode.

Please refer to the 'Common Programming Methods' section for the following:

- Stage Triggering Rung
- Stage String Output Rung
- Shutdown Trigger(s) Rung
- Non-Latched Output Control Rung(s)
- Next Stage Trigger Rung

RUNNING ► AUTO (Index: 6000)

This section governs operation of the system in **Auto Mode**, where sequences and functions are executed automatically based on predefined logic, interlocks, and process conditions. The system progresses through its operational stages without direct operator intervention, ensuring consistent, repeatable, and efficient performance. Centralising automatic control logic in a dedicated section improves readability, simplifies troubleshooting, and supports reliable long-term operation.

==== SEQUENCING [RUNNING - AUTO] ====

SECTION DESCRIPTION: The following section is for running the System in Auto Mode.

Please refer to the 'Common Programming Methods' section for the following:

- Stage Triggering Rung
- Stage String Output Rung
- Shutdown Trigger(s) Rung
- Non-Latched Output Control Rung(s)
- Next Stage Trigger Rung

WARM-DOWN (Index: 7000)

This section manages the controlled cool-down of system components following operation or prior to shutdown. It allows temperatures to reduce gradually using defined timing, airflow, or fluid circulation measures, preventing thermal shock and reducing mechanical stress. Implementing a dedicated cool-down phase improves equipment longevity, protects critical components, and ensures a safe transition to idle or shutdown states.

==== SEQUENCING [WARM-DOWN] ====

SECTION DESCRIPTION: The following section is for allowing system components to cool down under controlled measures.

Please refer to the 'Common Programming Methods' section for the following:

- Stage Triggering Rung
- Stage String Output Rung
- Shutdown Trigger(s) Rung
- Warm-Up / Warm-Down Trigger Rung
- Next Stage Trigger Rung

STOPPING (Index: 8000)

This section performs system or assembly post-run checks and manages the de-energisation of supplies following completion of operation. It verifies that equipment has stopped correctly, confirms safe conditions, and ensures that power and utilities are removed in a controlled manner. Centralising post-run validation and shutdown actions improves safety, supports reliable system reset, and prepares the system for idle, maintenance, or subsequent operation.

==== SEQUENCING [STOPPING] ====

SECTION DESCRIPTION: The following section is typically for completing System / Assembly Post Run Checks and or de-energizing supplies.

Please refer to the 'Common Programming Methods' section for the following:

- Stage Triggering Rung
- Stage String Output Rung
- Latched Output Control Rung(s)
- Next Stage Trigger Rung

STOPPED (Index: 9000)

This section completes system or assembly post-stopped checks to confirm that all equipment and processes have fully and safely come to rest. It verifies that motion has ceased, energy states are stable, and required safe conditions are met before allowing maintenance, reset, or restart. Centralising post-stopped validation improves safety, ensures clear state confirmation, and provides a reliable baseline for subsequent system actions.

==== SEQUENCING [STOPPED] ====

SECTION DESCRIPTION: The following section is typically for completing System / Assembly Post-Stopped Checks.

Please refer to the 'Common Programming Methods' section for the following:

- Stage Triggering Rung
- Stage String Output Rung
- Next Stage Trigger Rung

OPERATOR ATTENTION (Index: 9800)

This section is used to notify operators of tasks or actions that must be completed before the system can proceed with further operation. It provides clear prompts or messages identifying required interventions, acknowledgements, or checks, ensuring that outstanding conditions are addressed in a controlled manner. Centralising operator notifications improves communication, reduces ambiguity, and supports safe, coordinated interaction between personnel and the system.

==== SEQUENCING [OPERATOR ATTENTION] ====

SECTION DESCRIPTION: The following section is typically used for notifying the operators of tasks which need to be completed before further operation.

Please refer to the 'Common Programming Methods' section for the following:

- Stage Triggering Rung
- Stage String Output Rung
- Next Stage Trigger Rung

SYSTEM FAULT (Index: 9900)

This section is used to shut down equipment in response to detected faults and to inform operators or maintenance personnel of system or assembly issues. It manages fault-related shutdown actions and generates clear indications or messages identifying the cause of the shutdown, supporting rapid diagnosis and corrective action. Centralising fault shutdown and notification logic improves safety, ensures consistent system behaviour, and enhances maintainability and fault resolution efficiency.

==== SEQUENCING [SYSTEM FAULT] ====

SECTION DESCRIPTION: The following section is typically used for shutting equipment down and informing the operator / maintenance of System / Assembly faults.

Please refer to the 'Common Programming Methods' section for the following:

- Stage Triggering Rung
- Stage String Output Rung
- Latched Output Control Rung(s) (if required)
- Next Stage Trigger Rung

EMERGENCY STOP (Index: 9999)

This section handles the system response to an Emergency Stop event within a standard control PLC, managing the orderly shutdown of equipment and informing operators or maintenance personnel that an Emergency Stop has been triggered. It is important to note that this logic is **not a replacement for a certified safety system**; unlike safety-rated systems, a standard PLC does not provide the measurable certainty required for safety functions to operate when demanded. Certified safety systems—whether implemented using safety relays or safety PLCs—must be properly risk assessed and designed with appropriate control measures. The purpose of this section is limited to monitoring the Emergency Stop status for control and indication purposes only, such as disabling normal operation and providing clear HMI feedback, and does not constitute safety system design or certification guidance.

/// READ THE FOLLOWING MESSAGE ///

The following is **NOT** a replacement for a certified safety system. The key difference between a standard Programmable Logic Controller (PLC) and that of a safety system is the difference in measurable certainty. A safety system is designed to respond when required to. If you use a standard PLC for a safety system - you risk the system not operating when required. A safety system can be a simple of using Safety Relays or complicated using Safety PLCs. Regardless of the complexity - Safety Systems need to be correctly Risk Assessed and must appropriate Control Measures put in place. The following section is for the purpose programming a PLC of a standard control system responding to an Emergency Stop triggering (for example: having the Human Machine Interface (HMI) displaying that the Emergency System being triggered). This site does not provide any information with respect to Safety System design and certifications.

/// READ THE FOLLOWING MESSAGE ///**==== SEQUENCING [EMERGENCY STOP] ====**

SECTION DESCRIPTION: The following section is typically used for shutting equipment down and informing the operator / maintenance of System / Assembly Emergency Stop(s).

Please refer to the 'Common Programming Methods' section for the following:

- Stage Triggering Rung
- Stage String Output Rung
- Latched Output Control Rung(s) (if required)
- Next Stage Trigger Rung

Outputs Section

This section contains all output logic specific to the control of this program module, defining how actuators, devices, and signals are commanded during operation. Outputs that are shared or communal across multiple modules are intentionally excluded and managed within a separate input or common module to maintain clear ownership and separation of responsibility. This structure improves code clarity, reduces unintended interactions between modules, and simplifies maintenance and future expansion.

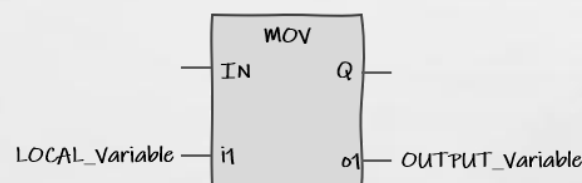
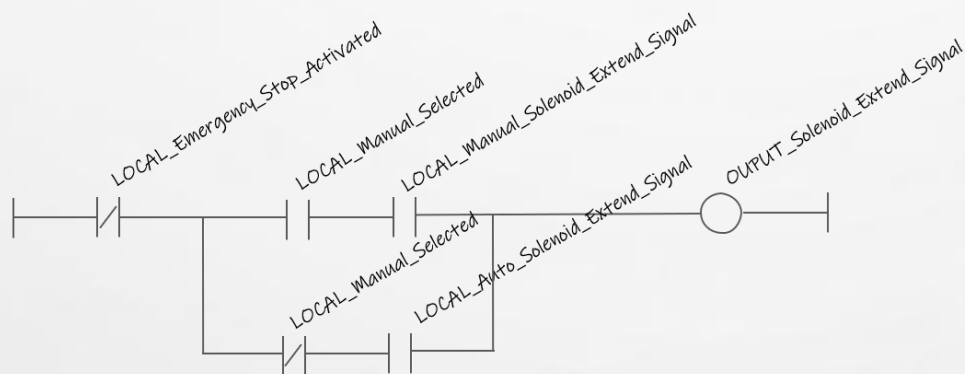
==== OUTPUTS ====

SECTION DESCRIPTION: The following section is for programming all outputs specific for the control of this program module. Any communal outputs should be included in a separate Input Module.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)

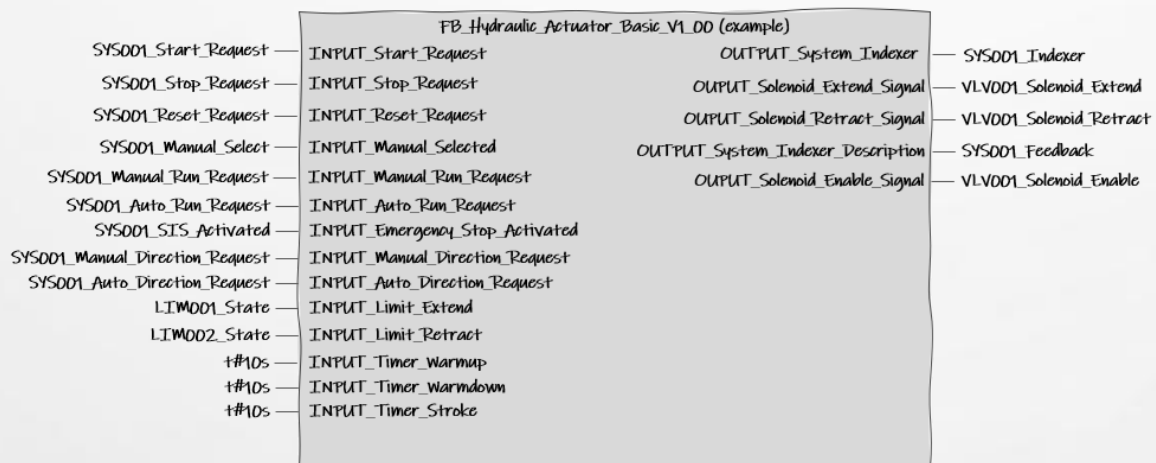


Programming Add-On (AOI) Sequence's

Add On Instructions (AOIs) refers to modules of code which are used repeatedly within a PLC. These can also be known as Function Blocks within some PLCs. An example of when a AOI may be used could be when programming a series of conveyors which are designed to run automatically. The programming for each physical section will be similar (like motor ramping, run-out timers etc) however the inputs and outputs will be different.

A programmer can use a AOI to save time writing specific code for each physical section of conveyor and program a module which can be deployed to multiple sections. This then standardises the functionality across all sections as well as making it easier to manage programming changes. Ideally, each AOI should be programmed in the same way as describe before using sequence programming. However, in some circumstances given the type of function (like indicator stack module) this will not be required.

System (SYSxxxx) Module(s)



Indicator Push Buttons Module(s)

This function block converts an indexed integer state variable into clear, discrete indicator outputs such as **Start**, **Stop**, and **Reset**, providing intuitive visual guidance for operators. By mapping the current system state to specific indicator or flashing outputs, the function ensures that the control interface clearly communicates which action is required for the system to proceed. This design improves operational understanding by making system expectations immediately visible—for example, highlighting or flashing the button that must be pressed next—thereby reducing operator uncertainty, preventing incorrect actions, and supporting safe, efficient interaction with the system.

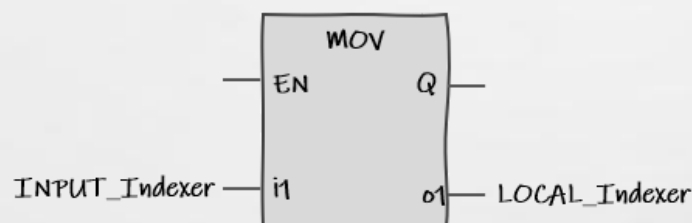
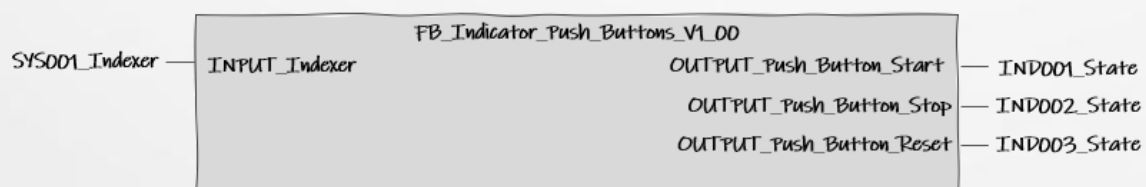
==== INDICATOR BUTTON TRANSLATION MODULE ====

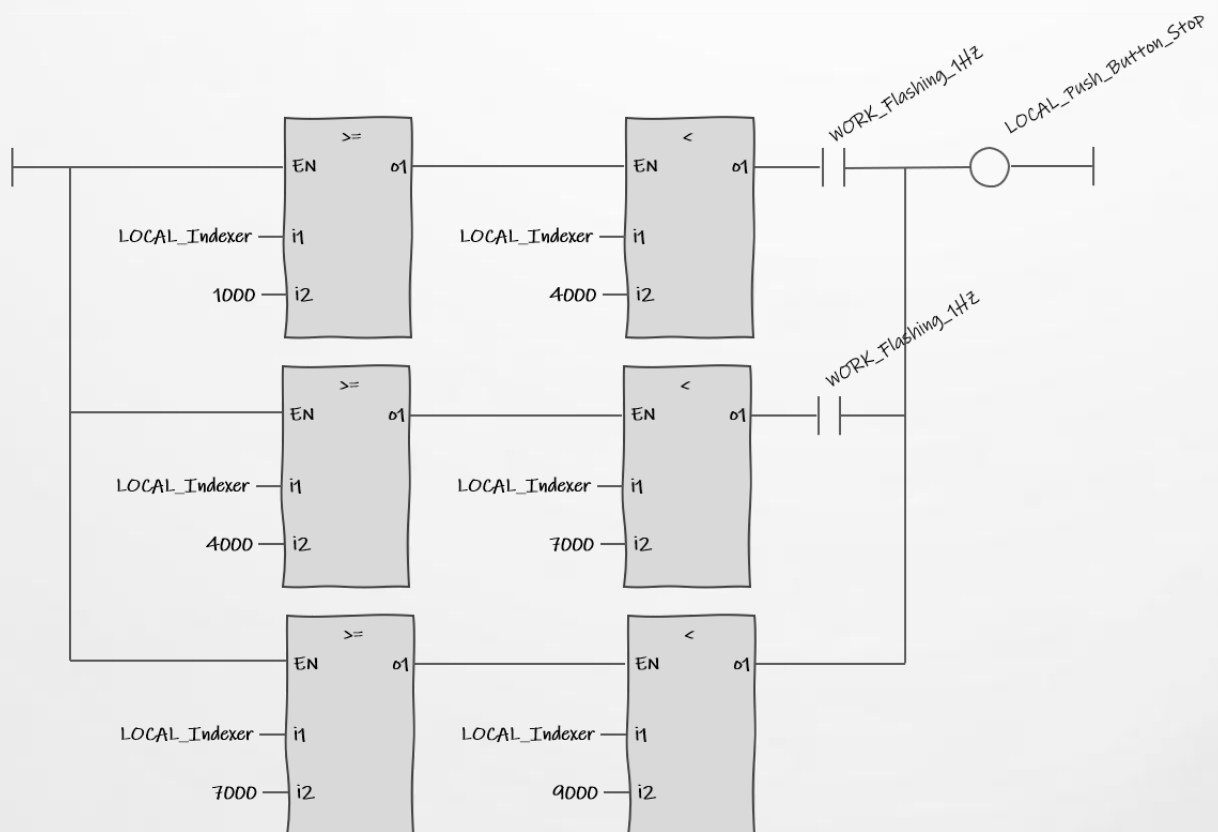
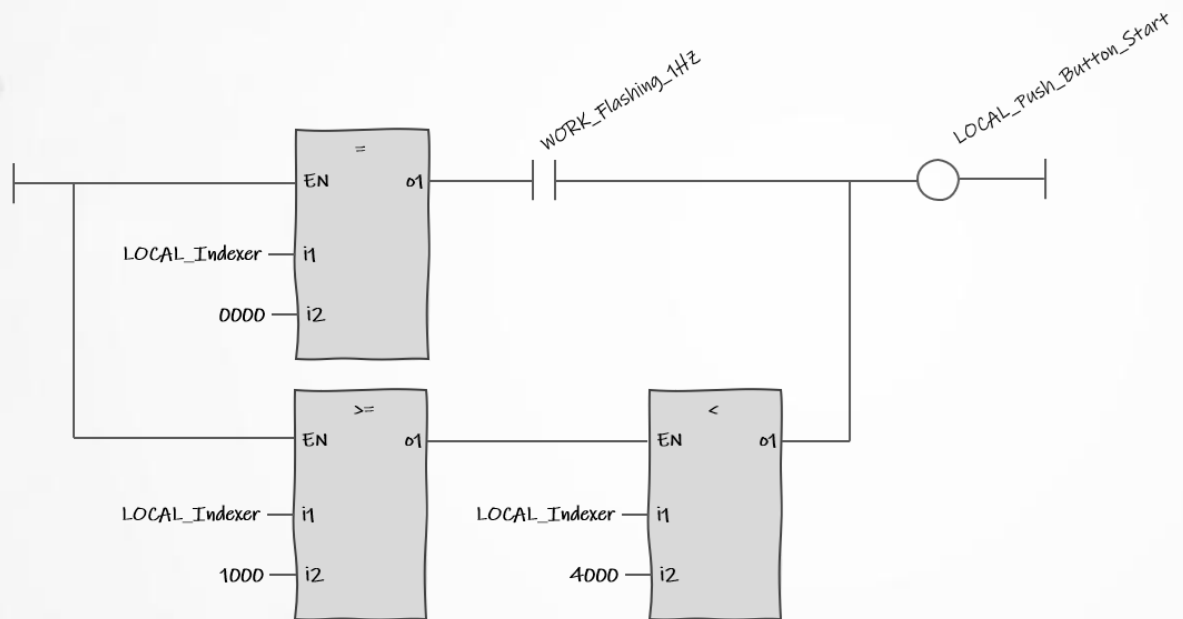
SECTION DESCRIPTION: The following mode is for converting a Sequence Indexer into outputs which are linked to Start / Stop / Reset Button Set

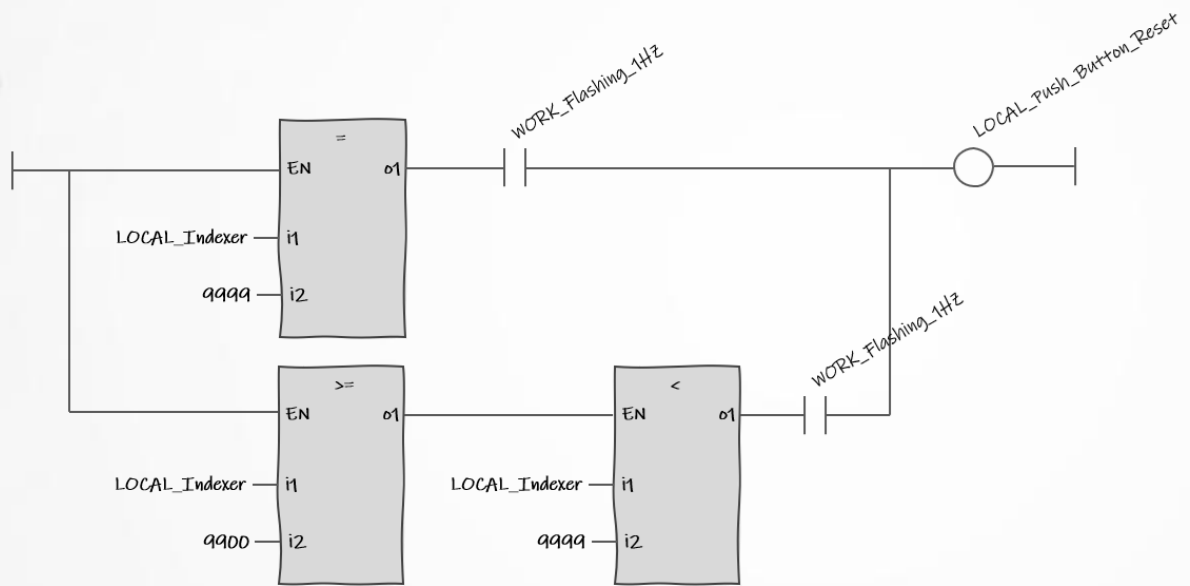
C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)







Indicator Stack Module(s)

This function manages the indicator stack to represent the current state of the system or assembly in a clear and consistent manner. Each indicator or light combination reflects a defined operating condition—such as idle, running, faulted, or stopped—providing an at-a-glance status rather than directing specific operator actions. By standardising how system states are displayed, the indicator stack improves situational awareness for operators and maintenance personnel, supports rapid assessment of system condition, and ensures consistent visual feedback across all operating modes.

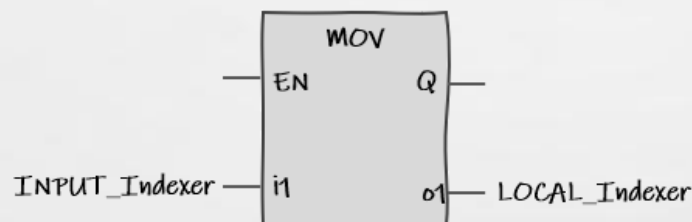
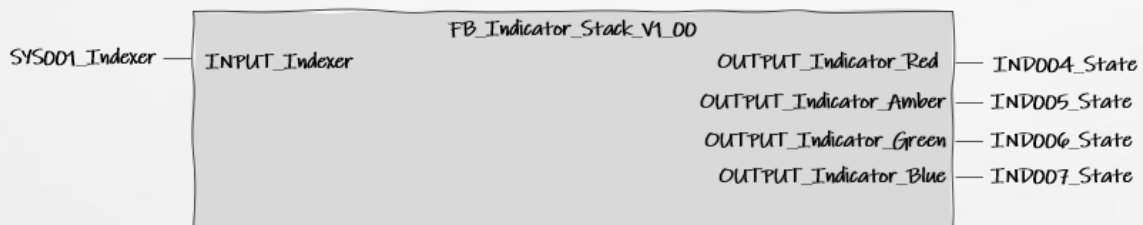
==== INDICATOR STACK TRANSLATION MODULE ====

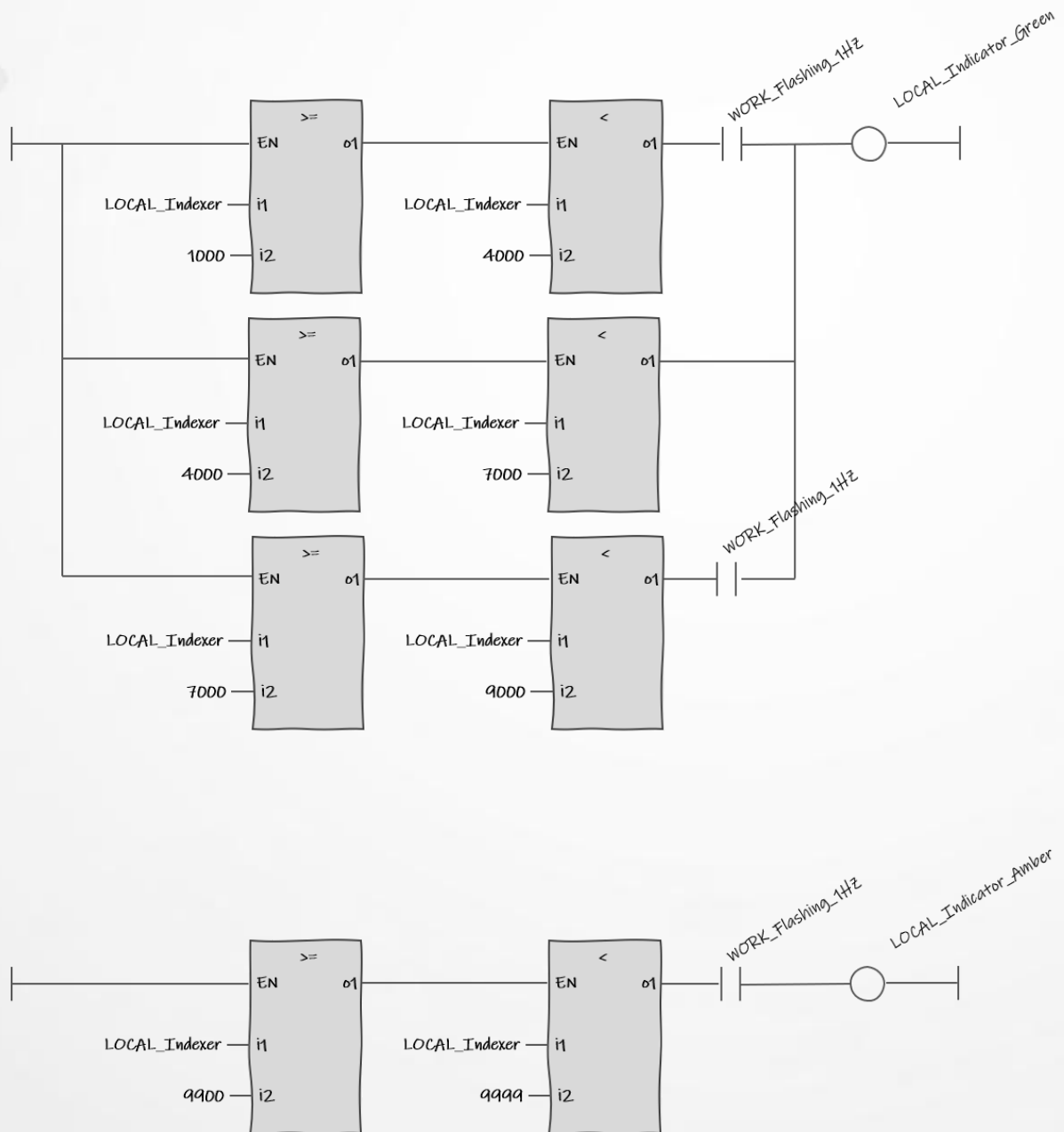
SECTION DESCRIPTION: The following mode is for converting a Sequence Indexer into outputs which are linked to an Indicator Stack.

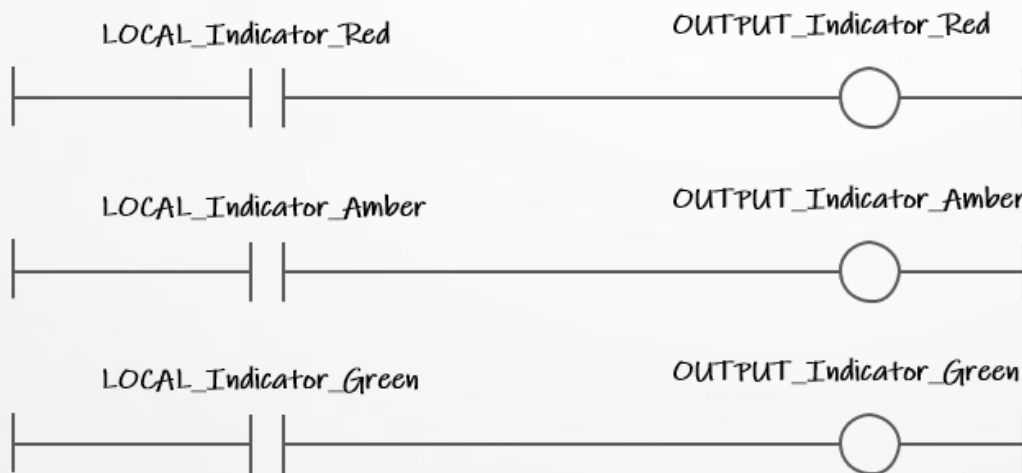
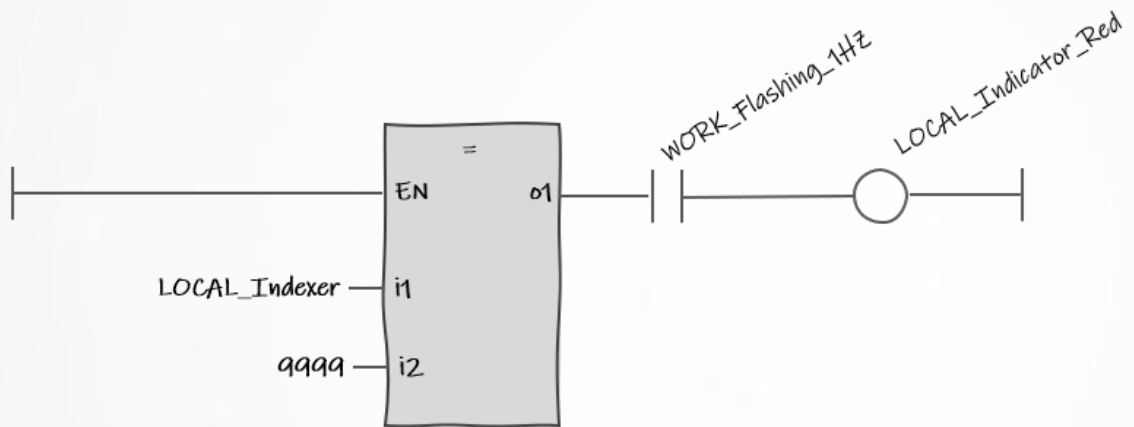
C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)







Common Programming Methods

The following functions are common utilities used across multiple stages of the program and are therefore documented in a central location for clarity and consistency. Centralising these shared functions and referencing them where required reduces duplication, simplifies maintenance, and makes the overall control philosophy easier to understand. This approach improves document readability, ensures consistent behaviour across stages, and allows updates to be managed efficiently in one place.

Stage Triggering Rung

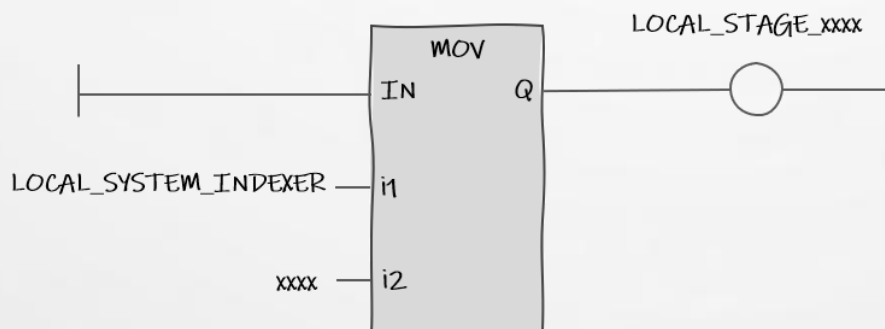
This logic evaluates the current index value and sets a corresponding stage-active flag to indicate which stage of the sequence is currently active. These flags provide a clear and consistent mechanism for enabling downstream logic, allowing subsequent rungs and functions to respond only when their associated stage is current. By centralising stage identification in this way, the sequence structure becomes easier to understand, troubleshoot, and modify, supporting predictable behaviour and maintainable program design.

RUNG DESCRIPTION: This logic checks the current index value and raises a stage-active flag, which is used by subsequent rungs to trigger logic when the corresponding stage is active.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



Stage String Output Rung

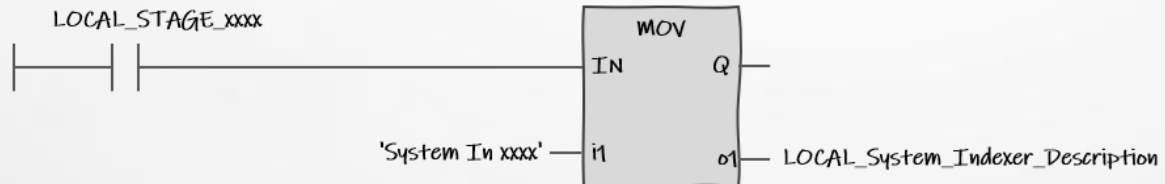
This rung assigns a status text string to an internal working variable, providing a consistent and centralised message for HMI operator feedback. By managing status messaging within the control logic, the HMI can display clear, standardised information that accurately reflects the current system state, improving operator awareness, reducing ambiguity, and supporting effective monitoring and troubleshooting.

RUNG DESCRIPTION: The following rung assigns a status text string to an internal working variable, providing a consistent message for HMI operator feedback.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



Shutdown Trigger(s) Rung

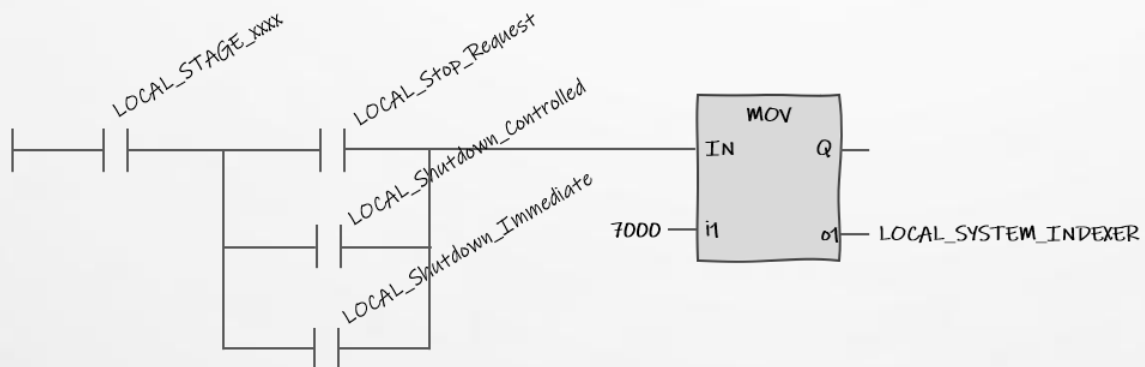
This rung forces the indexer directly into the shutdown phase when an immediate shutdown request is detected, intentionally bypassing any warm-down or controlled run-down logic. This ensures a rapid transition to a safe stopped state when conditions require urgent action, while maintaining a clear and deterministic sequence path for fault handling and system recovery.

RUNG DESCRIPTION: Forces the indexer to the shutdown phase on immediate shutdown request, bypassing warm-down logic.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



Latched Output Control Rung(s)

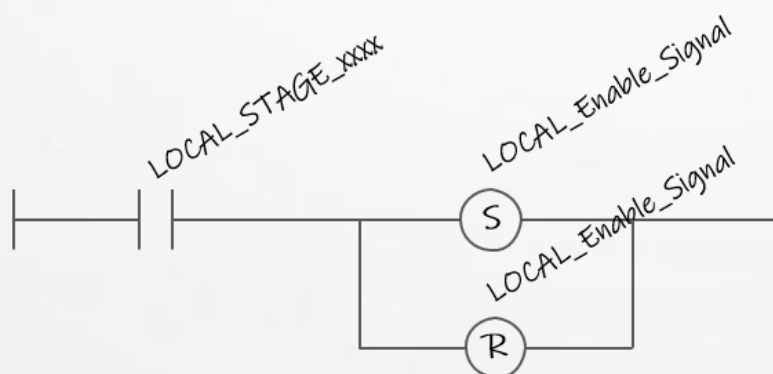
This rung activates and deactivates the required internal working bits for the current stage, which are then used to drive the associated output signals. By controlling outputs indirectly through stage-specific working bits, the logic remains clear, structured, and easy to maintain, ensuring consistent behaviour as the sequence progresses and simplifying future modifications or troubleshooting.

RUNG DESCRIPTION: The following rung activates / de-activates the required working bits for this stage, which in turn drive the associated output signals.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



Non-Latched Output Control Rung(s)

This rung implements scan-dependent, non-latched output control, where the output state directly follows the rung conditions on each PLC scan. The output is automatically de-energised when the enabling conditions are no longer true, ensuring responsive and deterministic behaviour. This approach is well suited to permissive-based control and interlocks, providing clear logic, predictable operation, and reduced risk of unintended output retention.

RUNG DESCRIPTION: Scan-dependent (non-latched) output control. Output follows rung conditions and resets automatically when conditions are no longer true.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



Warm-Up / Warm-Down Trigger Rung

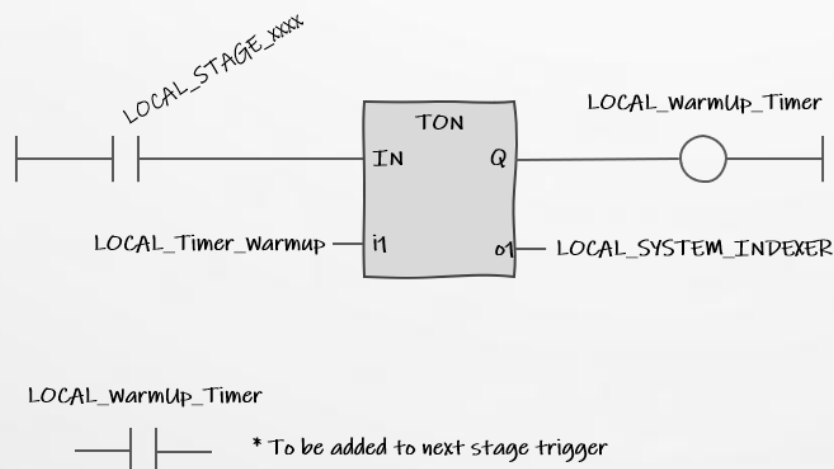
This rung provides stage timing by running a timer while the stage is active and using the timer's completion to trigger the transition to the next stage. By explicitly controlling dwell or delay times within each stage, the sequence progresses in a predictable and repeatable manner, ensuring that required conditions have sufficient time to be met before advancing and improving overall process stability and reliability.

RUNG DESCRIPTION: The following rung provides stage timing by running a timer during the active stage and using its completion to initiate the next stage transition.

C++ Example(s)

[Reserved for Future Use]

Ladder Example(s)



Next Stage Trigger Rung

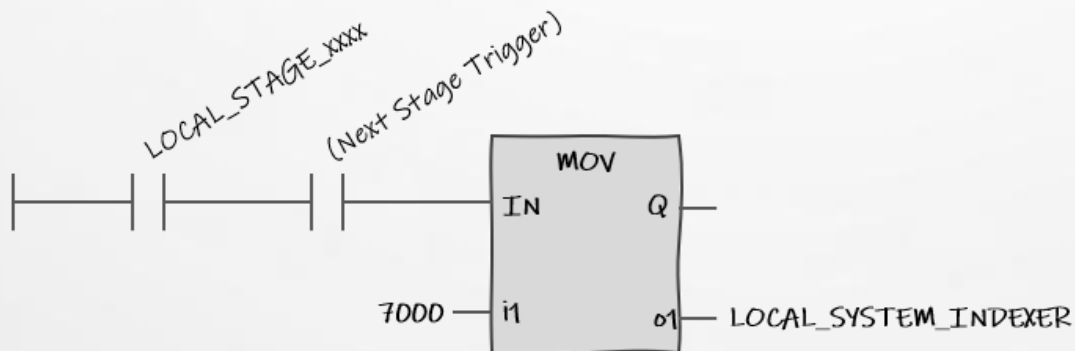
This rung controls stage progression by verifying that all required preconditions for the current stage have been satisfied before allowing the sequence to advance to the subsequent stage. By enforcing these checks, the logic ensures that each stage completes correctly and safely, preventing premature transitions, improving process reliability, and maintaining a clear, deterministic flow through the sequence.

RUNG DESCRIPTION: The following rung controls stage progression by verifying that all preconditions for the current stage have been fulfilled before stepping the sequence to the subsequent stage.

C++ Example(s)

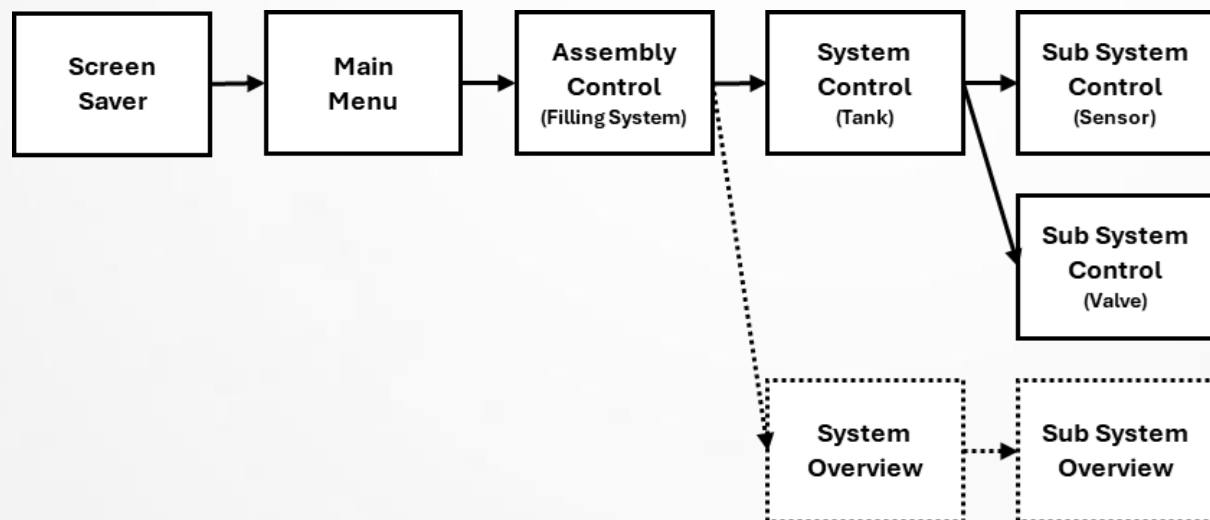
[Reserved for Future Use]

Ladder Example(s)



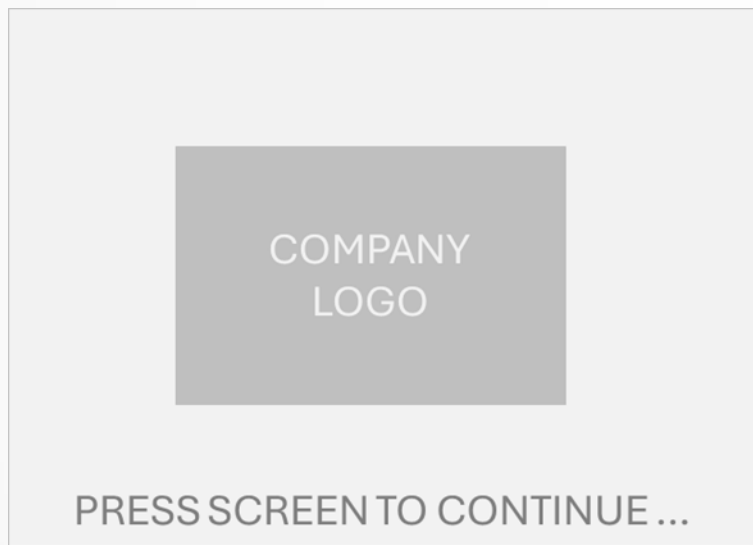
HMI PROGRAMMING

The aim of the HMI Programming is to provide an intuitive means of operation for operators and technicians. The System is programmed on the basis of controlling Sequences. When entering the Control Page of each Assembly / System, the operator will be able to START and STOP the Sequence. In the event that a sequence is not clear – a HELP screen can be provided detailing what conditions are expected and what should be occurring.



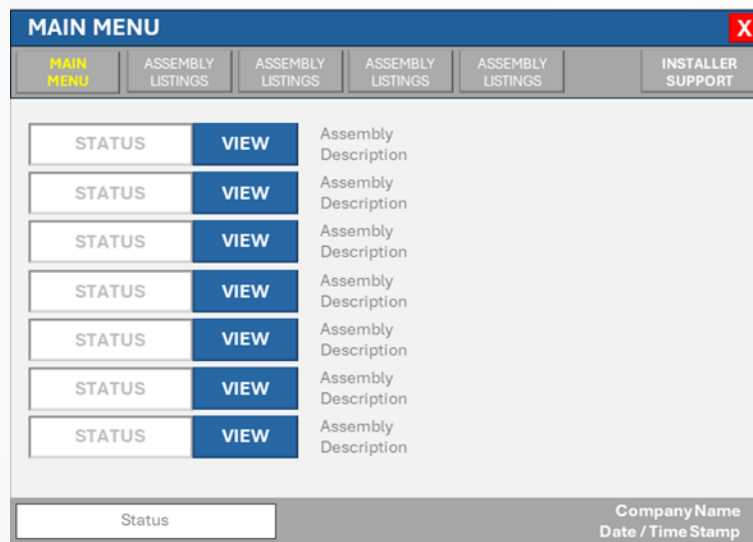
HMI's should be programmed in a way which is both functional and is intuitive. The aim of this standard is to design a HMI's interface which will allow operators to easily Start / Stop / Reset systems as well as allow maintenance staff to manually control and monitor Sub-Systems.

Screen Saver Screen



The following screen is a basic screen saver which has the logo of the company that owns the equipment. An invisible button component should be placed over the screen so that a user just needs to press the screen (anywhere) to progress to the next screen.

Main Menu Screen



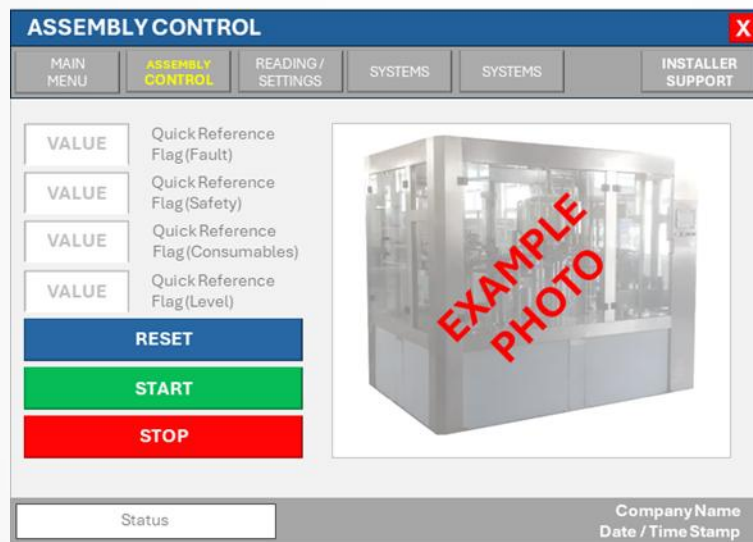
The screenshot shows a software interface titled "MAIN MENU" with a red close button in the top right corner. Below the title bar is a menu bar with six items: "MAIN MENU" (highlighted in yellow), "ASSEMBLY LISTINGS", "ASSEMBLY LISTINGS", "ASSEMBLY LISTINGS", "ASSEMBLY LISTINGS", and "INSTALLER SUPPORT". The main content area contains a table with seven rows. Each row has a "STATUS" column, a "VIEW" button, and a text column containing "Assembly Description". At the bottom of the screen, there is a status bar with a "Status" label on the left and "Company Name" and "Date / Time Stamp" on the right.

MAIN MENU		
×		
MAIN MENU ASSEMBLY LISTINGS ASSEMBLY LISTINGS ASSEMBLY LISTINGS ASSEMBLY LISTINGS INSTALLER SUPPORT		
STATUS	VIEW	Assembly Description
STATUS	VIEW	Assembly Description
STATUS	VIEW	Assembly Description
STATUS	VIEW	Assembly Description
STATUS	VIEW	Assembly Description
STATUS	VIEW	Assembly Description
STATUS	VIEW	Assembly Description

Status Company Name Date / Time Stamp

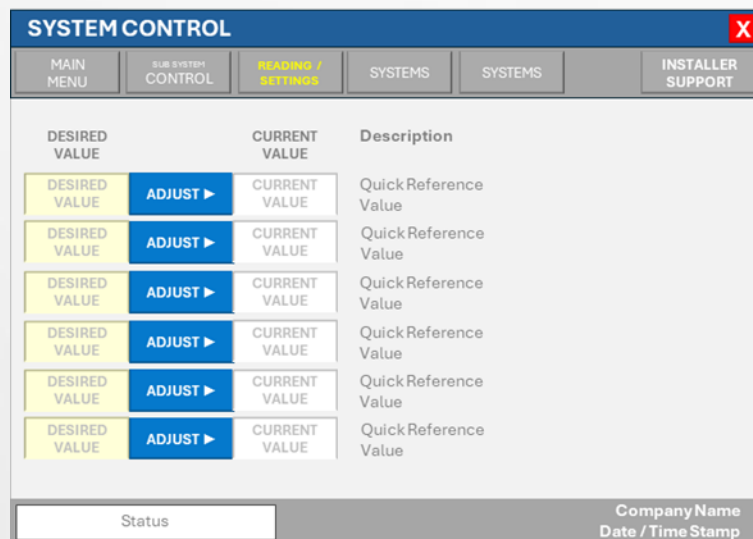
The following screen is for accessing all the difference Assembly's which the HMI is programmed to communicate with.

Assembly Control Screen



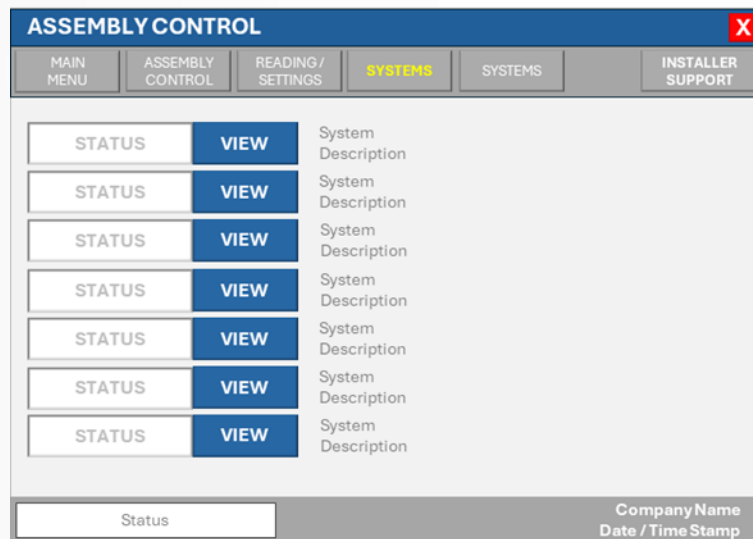
Access Level: Operators

The following screen provide quick reference values / flags which are typically required for the operators to monitor (Bins Full / Safety Tripped / Duct Blocked etc). This should have a basic Start / Stop / Reset control at the base of the page.



Access Level: Maintenance

The following screen allows for Assembly's set points to be programmed. An example of this could be the through-put counter which switches the Assembly in to Standby after a given number of cycles.



Access Level: Operators

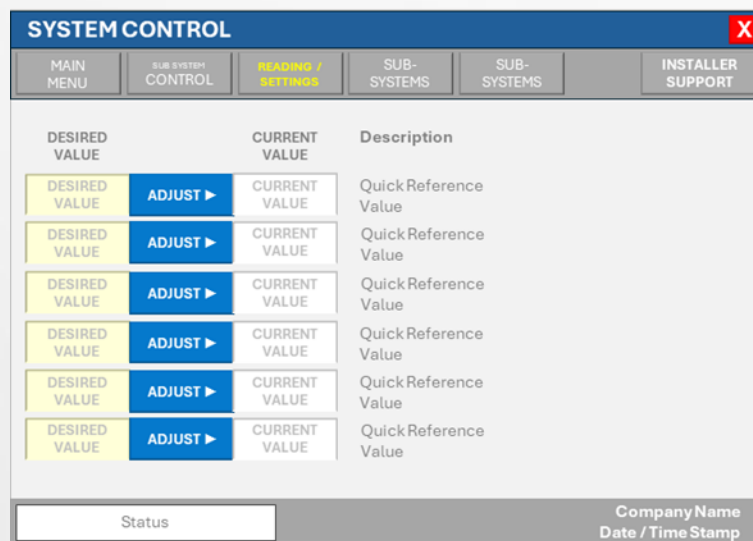
The following screen show System pages (these are those processes like Tank Level Management, which are connected to Sub Systems like level sensors and pumps). This page should also provide information regarding the status of every System.

System Control Screen



Access Level: Operators

Details: The following screen provides quick reference values / flags which are typically required for the operators to monitor. This should have a basic Start / Stop / Reset control at the base of the page.



Access Level: Maintenance

The following screen allows for System set points to be programmed into the system. An example of this could be the upper and lower levels of a storage water tank.

SYSTEM CONTROL					
MAIN MENU	SYSTEM CONTROL	READING / SETTINGS	SUB-SYSTEMS	SUB-SYSTEMS	INSTALLER SUPPORT
STATUS (IF APPLICABLE)	VIEW	Sub System Description			
STATUS (IF APPLICABLE)	VIEW	Sub System Description			
STATUS (IF APPLICABLE)	VIEW	Sub System Description			
STATUS (IF APPLICABLE)	VIEW	Sub System Description			
STATUS (IF APPLICABLE)	VIEW	Sub System Description			
STATUS (IF APPLICABLE)	VIEW	Sub System Description			
STATUS (IF APPLICABLE)	VIEW	Sub System Description			

Status

Company Name
Date / Time Stamp

Access Level: Operators

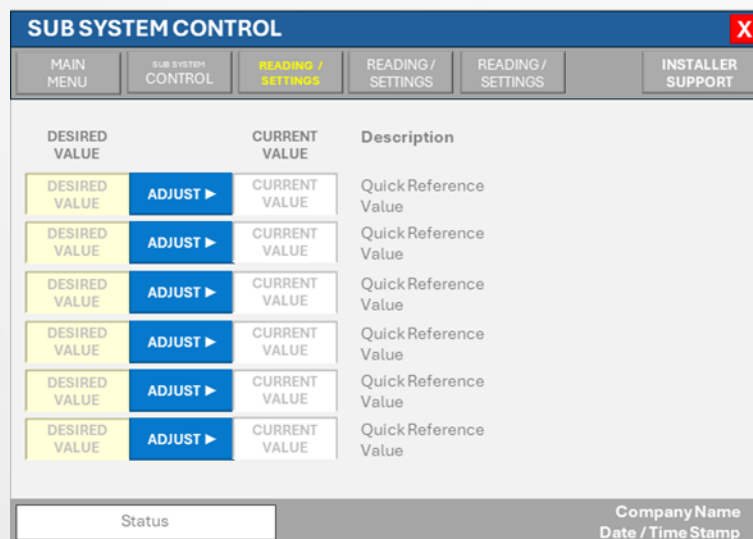
The following screen shows Sub-System pages. These are typically the individual components (for example sensors, VSDs, etc)

Sub System Control Screen



Access Level: Operators

The following screen provide quick reference values / flags which are typically required for the operators to monitor. There should be no Start / Stop / Reset function at this level.



Access Level: Maintenance

This screen should provide feedback and settings which are programmable by the sensor / device connected. An example of this maybe a current reading of a sensor or the adjustment of a muting distance of a sensor.

Standard Colours

Title Bar#215F99

Menu Bar:.....#A6A6A6

Selected Item:#A6A6A6

Background:.....#F2F2F2

Reset button:.....#215F99

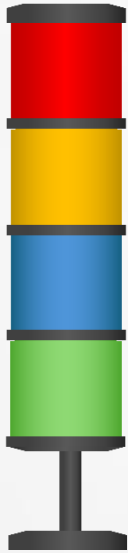
Start button: #05BC56

Stop button:.....#FF0506

Footer Bar:#A6A6A6

Installing Hardware

Indicator Stack Configuration



The light stack is recommended to be installed in the following manner. Given that the Red Indicator is typically setup as a Emergency Type indicator which requires immediate attention – it is held to the highest vertical position. Subsequently, the next prioritised stage is that of any maintenance issues represented by the Orange Indicator. Both of these will shut the Assembly / System down if they were to occur.

The next step down if the Green Indicator which indicates that the system is Running. This allows operators and Team Leaders to clearly identify that systems are within acceptable ranges from a distance. The last indicator being Blue, is typically used for operator invention (for example re-stocking consumables).

It is recommended that the urgency of the indicator be made from most urgent and critical being at the top – to least critical being at the bottom.

Assembly Stack Red Indicator

	Assembly Red Indicator Stack OFF Used to indicate that there are no Emergency Triggers, and that the system is in a work safe state. Presented when the Assembly / System is in one of the following stage(s); • All other stages
	Assembly Red Indicator Stack FLASHING (1Hz) Used to indicate that an Emergency Trigger has occurred and requires immediate attention. Presented when the Assembly / System is in one of the following stage(s); • Emergency
	Assembly Red Indicator Stack SOLID ON Not used.

Assembly Stack Orange Indicator

	Assembly Orange Indicator Stack OFF Used to indicate that there are no Faults present and that the system is in a serviceable state. Presented when the Assembly / System is in one of the following stage(s);
	Assembly Orange Indicator Stack FLASHING (1Hz) Used to indicate that a Fault has occurred and requires attention. Presented when the Assembly / System is in one of the following stage(s); • System Fault
	Assembly Orange Indicator Stack SOLID ON Not used.

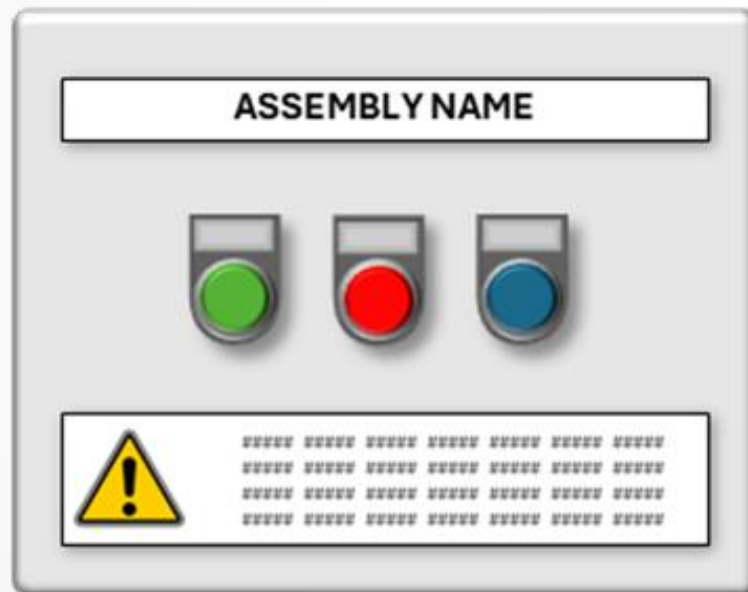
Assembly Stack Blue Indicator

	Assembly Blue Indicator Stack OFF Typically used in indicating that no operator attention is required. This indicator in this state indicates that there are no operate type interventions required.
	Assembly Blue Indicator Stack FLASHING (1Hz) Typically used to indicate that the Assembly / System required operator attention. This is used to indicate that a operate duty is due shortly.
	Assembly Blue Indicator Stack SOLID ON Typically used to indicate that the Assembly / System required immediate operator attention. This is used to indicate that a operate duty is required before starting the Assembly / System again.

Assembly Stack Green Indicator

	Assembly Green Indicator Stack OFF Used to indicate that the Assembly is no Running. Presented when the Assembly / System is in one of the following stage(s); • All other stages
	Assembly Green Indicator Stack FLASHING (1Hz) Used to indicate that the Assembly / System is in the process of either starting or stopping. Typically used in a change of operational state. Presented when the Assembly / System is in one of the following stage(s); • Starting • Started • Warm-up • Warm-down • Stopping
	Assembly Green Indicator Stack SOLID ON ... Presented when the Assembly / System is in one of the following stage(s); • Running (All)

Operator Control Panel



The following configuration is recommended for Push Button control. This configuration can be installed on a dedicated System (focusing on a single process like monitoring the liquid level of a tank) or for the entire Assembly.

To make the system intuitive and easy for upgrading, the button configuration matches what is recommended within an HMI.

This can be installed on the control board, which houses the PLC controlling the Assembly, or on its own operator panel. It is highly recommended that any controls, like that pictured above, are installed in a location that is in viewing distance of the machinery in which it controls.

Assembly / System Start Button / Indicator

	Assembly Start Button / Indicator OFF Presented when the Assembly / System is Stopped and not ready for Start-up (in fault / emergency condition). Presented when the Assembly / System is in one of the following stage(s); • Warm-down • Stopping • Stopped • Fault • Emergency
	Assembly Start Button / Indicator FLASHING (1Hz) Presented when the Assembly / System is ready for Starting. Indicator flashing in attempt to gain the operators attention. Presented when the Assembly / System is in one of the following stage(s); • Standby
	Assembly Start Button / Indicator SOLID ON Presented when the Assembly / System is Running. Presented when the Assembly / System is in one of the following stage(s); • Start-up • Started • Warm-up • Running

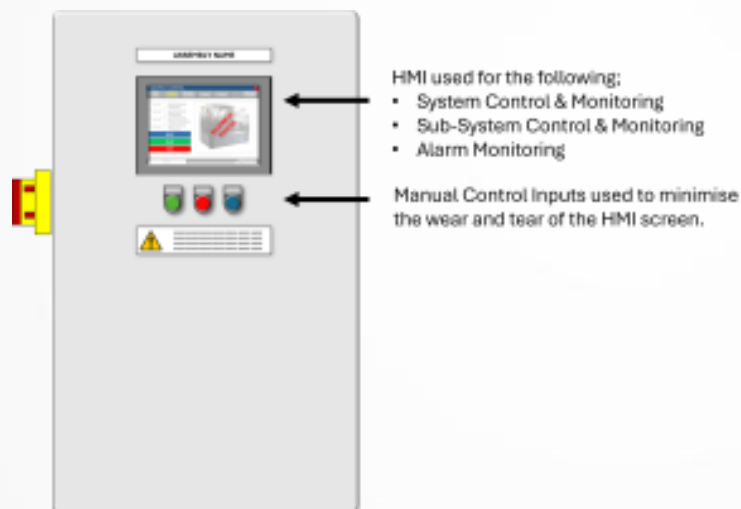
Assembly / System Stop Button / Indicator

	Assembly Stop Button / Indicator OFF Presented when the Assembly / System is Started. Presented when the Assembly / System is in one of the following stage(s); • Start-up • Started • Warm-up
	Assembly Stop Button / Indicator FLASHING (1Hz) Presented when the Assembly / System is ready for Shut-down. Indicator flashing in attempt to gain the operators attention. Presented when the Assembly / System is in one of the following stage(s); • Start-up • Started • Warm-up • Running
	Assembly Stop Button / Indicator SOLID ON Presented when the Assembly / System is Stopped / Standby. Presented when the Assembly / System is in one of the following stage(s); • Warm-up Down • Stopping • Stopped • Standby • Fault • Emergency

Assembly / System Reset Button / Indicator

	Assembly Reset Button / Indicator OFF Present when there are no faults present in the Assembly / System. Presented when the Assembly / System is in one of the following stage(s); • Standby • Starting • Started • Warm-up • Running (All) • Warm-down • Stopping • Stopped • Standby
	Assembly Reset Button / Indicator FLASHING (1Hz) Present when there is a Resettable fault within the Assembly / System. Presented when the Assembly / System is in one of the following stage(s); • Fault
	Assembly Reset Button / Indicator SOLID ON Present when there is a Non-Resettable fault (from the perspective that the Safety System is independent to the Control PLC) within the Assembly / System. Presented when the Assembly / System is in one of the following stage(s); • Emergency

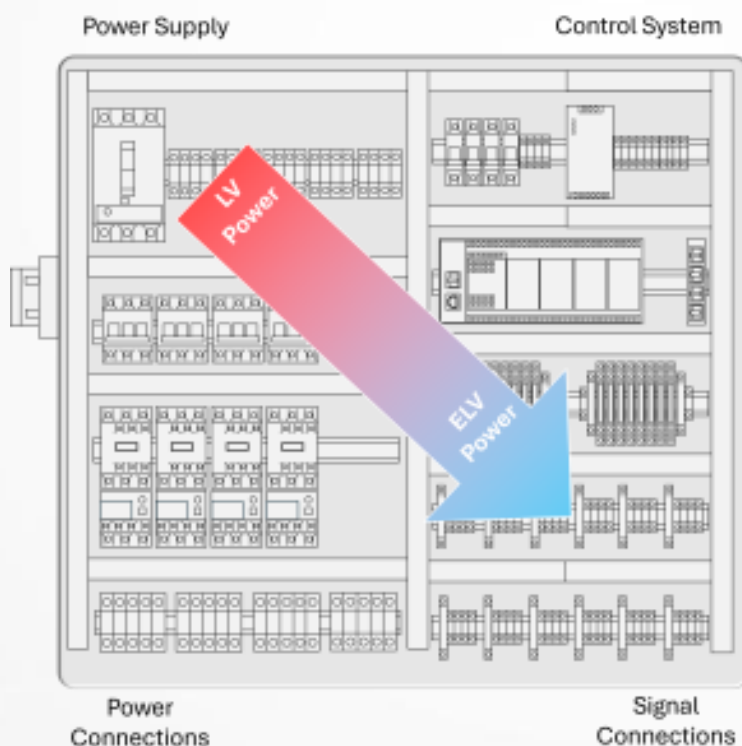
Main Control Panel External Layout



Standardising the control panel layout provides significant benefits in safety, usability, and operational consistency. Keeping the front-of-panel controls simple—typically limited to **Start**, **Stop**, and **Reset**—ensures that operators can quickly and confidently interact with the system without confusion, regardless of which panel they are working on. This approach reduces training requirements and the risk of incorrect operation. Physical manual controls should only be added where genuinely required, such as when risk assessments determine that reliance on the HMI is unacceptable due to availability or safety concerns. In most cases, complex or mode-specific controls are better managed through the HMI, where context, interlocks, and feedback can be clearly presented.

Equally important is the consistent placement of supporting features on the panel exterior. Clear system warning labels on the front of the panel provide immediate hazard awareness before access or operation, supporting safe working practices. A locally mounted isolator on the side of the panel allows safe and convenient electrical isolation during maintenance or emergency situations without opening the enclosure. Including a clearly visible panel ID on the front of the panel ensures unambiguous identification for operators, maintenance personnel, and documentation reference, particularly in multi-panel installations. Together, these standardisation principles improve safety, reduce errors, and support efficient operation and maintenance across the system lifecycle.

Power & Voltage Pathways

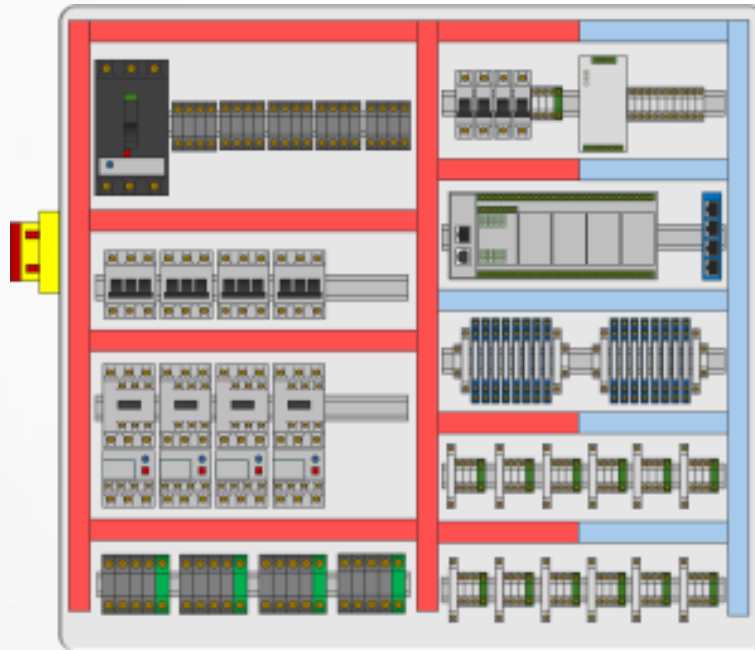


Applying a defined Low Voltage (LV) and Extra Low Voltage (ELV) layout within a switchboard delivers clear safety and operational advantages. Positioning LV components toward the upper left of the board and ELV components toward the lower right—effectively dividing the switchboard into functional halves as shown—creates clear electrical segregation. This reduces the risk of accidental contact, electromagnetic interference, and fault propagation from higher-energy LV circuits into sensitive ELV control and communication systems, improving overall safety and reliability.

This structured layout also improves maintainability and future flexibility. Technicians can quickly understand the board's organization, making installation, commissioning, and fault-finding faster and less error-prone. Consolidating ELV equipment in a dedicated lower-right zone provides a safe, accessible area for control and instrumentation, while the LV section supports efficient power distribution and thermal management. The result is a switchboard that is safer to work on, easier to expand, and aligned with best-practice electrical design principles.

Component Layout

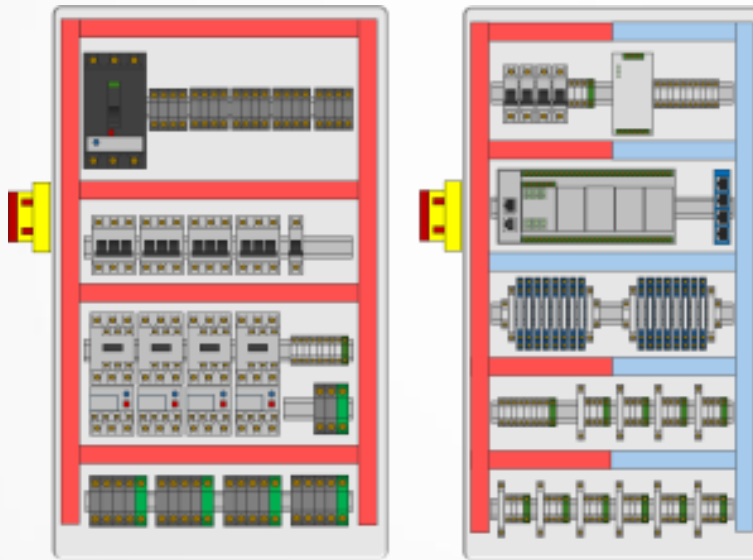
Combined Power and Control System



Separating Low Voltage (LV) and Extra Low Voltage (ELV) circuits within a single control panel provides a balanced solution where space, cost, or footprint constraints apply. By clearly zoning LV power components and ELV control and instrumentation equipment within the same enclosure, effective electrical segregation is achieved without sacrificing accessibility. This approach reduces the risk of electrical interference, improves personnel safety during maintenance, and ensures sensitive ELV circuits are protected from induced voltages and thermal effects associated with LV equipment.

This integrated layout also supports efficient installation and servicing by maintaining a clear, logical structure that is easy to interpret. Technicians can work on ELV circuits with reduced exposure to LV components, while future modifications can be accommodated within defined zones. When properly designed, a combined panel meets segregation requirements, maintains compliance with standards, and delivers a practical, space-efficient control solution.

Separated Power and Control System

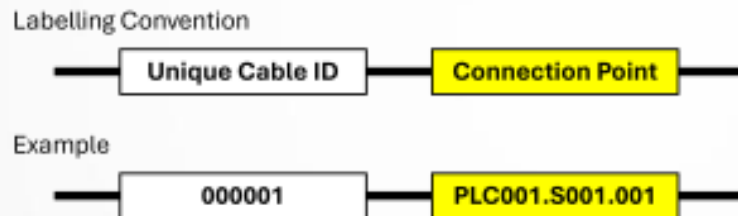


Using separate cabinets for LV and ELV circuits represents the highest level of segregation and is ideal for complex or safety-critical systems. Housing LV power equipment in one panel and ELV control, automation, and communication devices in another virtually eliminates the risk of electrical interference, accidental contact, or fault transfer between voltage classes. This physical separation significantly enhances personnel safety and protects sensitive control equipment from electrical noise and thermal stress.

A dual-panel arrangement also improves scalability and lifecycle management. ELV control panels can be expanded or modified independently of LV power infrastructure, reducing downtime and simplifying upgrades. Maintenance activities are safer and more efficient, as technicians can work within the ELV cabinet without exposure to LV hazards. This approach demonstrates strong engineering discipline, delivers long-term flexibility, and aligns with best-practice design for robust and future-ready control systems.

Connection Configuration(s)

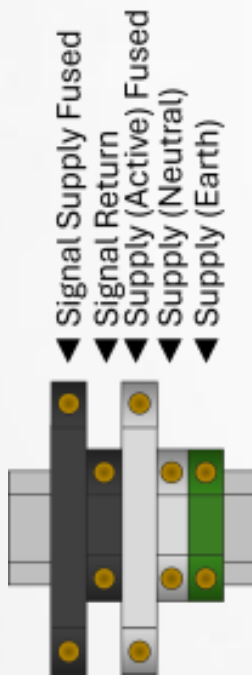
Cable Labelling



Applying a structured cable labelling system with unique cable identifiers and clear connection labels significantly improves safety, accuracy, and efficiency across installation and maintenance activities. Assigning each cable a unique cable number, displayed on yellow identification labels, provides immediate traceability from drawings to physical installation. This allows technicians to quickly verify circuits, reduces the risk of cross-connection, and minimizes errors during commissioning or fault-finding, particularly in dense control panels or cable routes.

In addition, using yellow connection labels to clearly identify termination points—indicating what each cable is connected to—enhances clarity at both ends of the cable. This dual-labelling approach ensures that cables can be positively identified without reliance on assumptions or manual tracing, reducing downtime and improving workplace safety. A consistent yellow labelling standard makes identification instantly recognizable, supports compliance with good engineering practice, and delivers long-term benefits by simplifying modifications, inspections, and future expansions of the electrical system.

Sensor Connection(s)



Implementing a standardised terminal layout for cables entering and exiting a cabinet delivers clear benefits in safety, consistency, and ease of maintenance. Using single-level (non-stacked) terminals ensures that each connection point is fully visible and accessible, reducing the risk of wiring errors and simplifying inspection and fault-finding. Establishing a consistent terminal order—power supply first (with the positive conductor fused), followed by power return, signal supply and return, and finally earth—creates a logical and repeatable structure that is immediately recognisable to installers and technicians.

Grouping all associated terminals in close proximity further improves installation quality and long-term reliability. By allowing conductors from a single cable sheath to terminate adjacent to one another, the outer sheath can be maintained as close as possible to the termination point, improving mechanical protection, strain relief, and EMC performance.

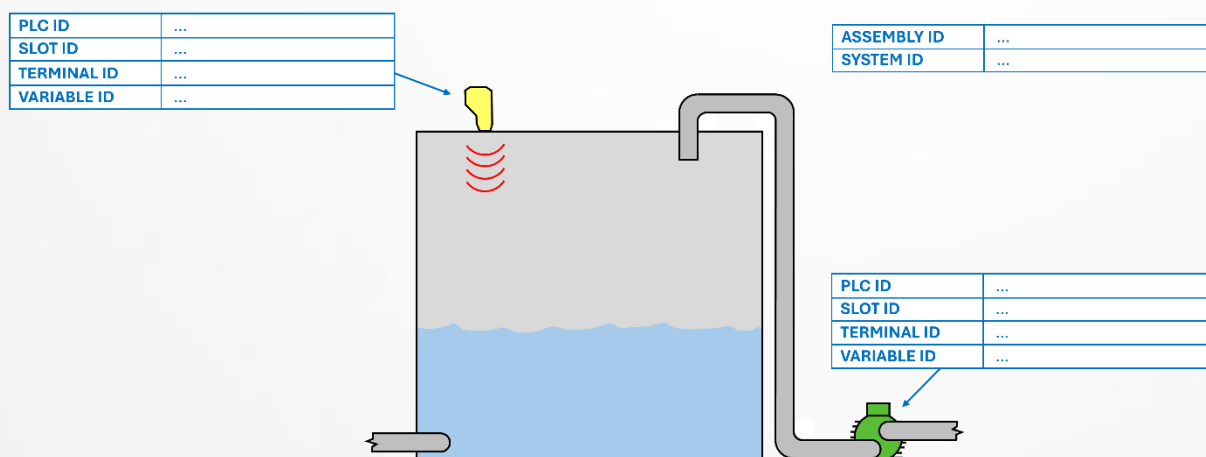
This approach results in cleaner wiring, reduced conductor stress, and a more professional finish, while also supporting safer modifications and future expansion through a clear, standardised terminal philosophy.

Documentation Overview

Schematic Legend / Title

LOGO	DRAWN BY	...	SITE	...		
	DRAWN DATE	...	ASSEMBLY	...		
	...	...	PROJECT	...		
	...	...	CURRENT PAGE	...	NEXT PAGE	...

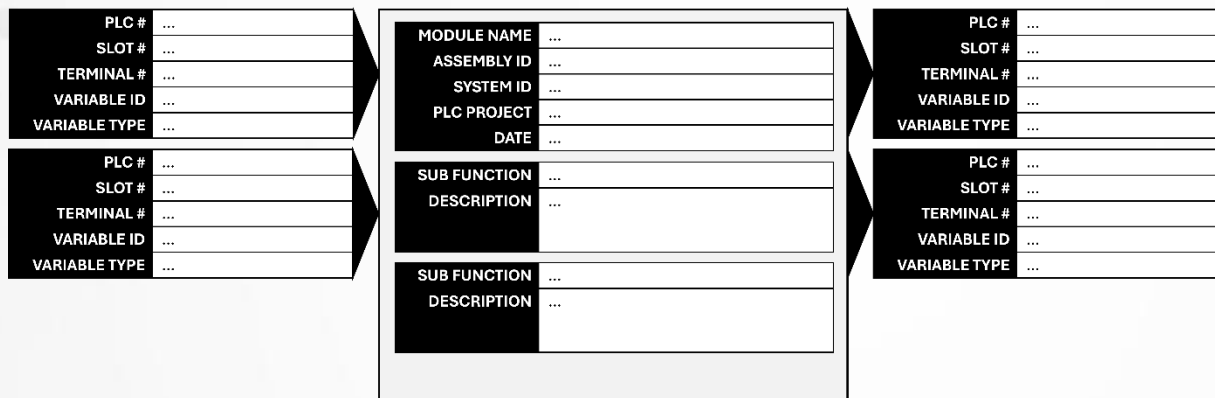
System Overview Page



Providing a dedicated system overview diagram that clearly identifies all subsystems, equipment names, unique IDs, PLC connection points, and associated terminal references delivers significant benefits during commissioning, operation, and fault-finding. By consolidating this information into a single, easy-to-read reference, engineers and technicians can quickly understand system structure and signal flow without needing to navigate multiple drawings or documents. This reduces troubleshooting time, minimizes the risk of misidentification, and supports safer, more confident decision-making when diagnosing faults.

Replicating this system diagram within the HMI further enhances usability and long-term value. A consistent visual representation between documentation and operator interfaces allows personnel to correlate physical hardware, PLC logic, and live system status at a glance. This shared reference improves knowledge transfer, simplifies future modifications, and ensures that changes can be implemented and verified efficiently. Overall, a standardised system diagram promotes clarity, maintainability, and lifecycle resilience across both engineering and operational disciplines.

Programming Overview Page



A **Functional I/O Interface Diagram** provides a clear, structured overview of how each system or subsystem function is connected to its associated inputs and outputs. The diagram consolidates critical information such as subsystem names and IDs, PLC references (rack, slot, and channel), terminal numbers, variable identifiers, and signal types into a single, easy-to-read format. By presenting both hardware and logical relationships together, it allows engineers and technicians to quickly understand how a function is implemented, regardless of whether the signal originates from field devices, hardwired logic, or PLC-based control.

The primary benefit of this diagram is improved efficiency and accuracy during commissioning, fault-finding, and future system modifications. It significantly reduces troubleshooting time by providing a single point of reference, eliminating the need to cross-check multiple drawings, I/O lists, and PLC programs. When replicated within HMIs, the diagram further enhances system transparency by allowing live signal status to be viewed in context, supporting faster diagnosis and safer intervention. Overall, the Functional I/O Interface Diagram improves system clarity, knowledge transfer, and long-term maintainability across the full lifecycle of the installation.

Summary of Recommended Design Principles

This design philosophy is presented as an open-source framework intended to encourage consistent, safe, and maintainable control system design. The principles outlined focus on reducing unnecessary complexity, improving clarity of operation, and promoting a shared understanding across operators, engineers, and maintenance personnel. By applying standardised layouts, structured programming practices, and clear operator feedback, systems become easier to commission, fault-find, modify, and operate throughout their lifecycle.

While these guidelines are not prescriptive or restrictive, they are intended to support informed engineering judgement and site-specific risk assessment. Adopting a common approach enables better knowledge transfer, reduces reliance on individual familiarity, and improves overall system reliability. Ultimately, this philosophy aims to create control systems that are intuitive to use, straightforward to maintain, and resilient to change—benefiting all stakeholders involved in the operation and support of the system.